UNIVERSIDAD COMPLUTENSE DE MADRID

FACULTAD DE INFORMÁTICA
Departamento de Arquitectura de Computadores y Automática

TESIS DOCTORAL

Técnicas de gestión de infraestructuras virtuales en entornos multi-cloud

Virtual infrastructures management techniques in multi-cloud environments

MEMORIA PARA OPTAR AL GRADO DE DOCTOR

PRESENTADA POR

José Luis Lucas Simarro

Directores

Rafael Aurelio Moreno Vozmediano
Rubén Manuel Santiago Montero

Madrid, 2016

Técnicas de Gestión de Infraestructuras Virtuales en Entornos Multi-Cloud

Virtual Infrastructures Management Techniques in Multi-Cloud Environments

Tesis Doctoral

Jose Luis Lucas Simarro

Departamento de Arquitectura de Computadores y Automática
Facultad de Informática
Universidad Complutense de Madrid

Octubre, 2015

Técnicas de Gestión de Infraestructuras Virtuales en Entornos Multi-Cloud

Virtual Infrastructures Management Techniques in Multi-Cloud Environments

Tesis presentada por

Jose Luis Lucas Simarro

para optar al título de Doctor en Informática

—

Tesis dirigida por los Doctores

D. Rafael Moreno-Vozmediano y D. Ruben S. Montero

Departamento de Arquitectura de Computadores y Automática

Facultad de Informática

Universidad Complutense de Madrid

Octubre, 2015

Técnicas de Gestión de Infraestructuras Virtuales en Entornos Multi-Cloud

Memoria presentada por Jose Luis Lucas Simarro para optar al grado de Doctor por la Universidad Complutense de Madrid, realizada bajo la dirección de D. Rafael Moreno-Vozmediano y D. Ruben S. Montero (Departamento de Arquitectura de Computadores y Automática, Universidad Complutense de Madrid).

Virtual Infrastructures Management Techniques in Multi-Cloud Environments

Report presented by Jose Luis Lucas Simarro to the Complutense University of Madrid in order to apply for the Doctor's degree. This work has been supervised by D. Rafael Moreno-Vozmediano y D. Ruben S. Montero (Computers Architecture and Automation Department, Complutense University of Madrid).

Madrid, Octubre 2015

The research leading to these results has received funding from the European Union's 7th Framework Programme ([FP7/2007-2013]) under grant agreement no 258862 (4CaaSt); from Consejería de Educación of Comunidad de Madrid, Fondo Europeo de Desarrollo Regional, and Fondo Social Europeo through MEDIANET Research Program S2009/TIC-1468; and from Ministerio de Ciencia e Innovación of Spain through research grants TIN2009-07146 (HPCCloud) and TIN2012-31518 (ServiceCloud).

A vosotros, que creísteis en mi
y me apoyasteis de cualquier forma
durante esta aventura,
a vosotros os dedico esta tesis,
porque vosotros sois, en vuestra proporción,
tan merecedores de esto como yo.

Agradecimientos

No hay inversión más rentable que la del conocimiento.

Benjamin Franklin

Acabar una tesis requiere mucho trabajo y mucho esfuerzo, lo que inevitablemete lleva consigo mucho tiempo. Durante este tiempo me he dado cuenta de la importancia que ha tenido cierta gente de mi alrededor para que esta tesis llegue a buen puerto. Al final es cuando tengo la oportunidad de agradecer lo mucho que han hecho por mi, y por lo tanto, aquí van mis agradecimientos.

En primer lugar, me gustaría agradecer la oportunidad que tuve de empezar esta aventura a quienes lo hicieron posible. Gracias a Blanca Caminero y Carmen Carrión, mis directoras del Master, por permitirme empezar en el mundillo de la investigación en mi época de Albacete. Y gracias a Nacho Martín, a Ruben Santiago y a Rafael Moreno, por aceptarme como doctorando, por incluirme en el grupo de investigación DSA (Distributed System Architectures), y por concederme la beca FPI, que me aseguró cuatro años de sustento económico para seguir con la investigación y con mi vida personal aquí en Madrid.

En especial, quiero agradecer a Rafael Moreno y a Ruben Santiago, mis directores de tesis, por su trabajo durante todo este tiempo. Para empezar y acabar una tesis, es fundamental la figura de los directores de tesis, aquellos que, al principio de la aventura, te muestran el camino a seguir, te facilitan el trabajo en lo posible, o te ayudan con la redacción del primer artículo científico. Durante la tesis, son fundamentales sus revisiones técnicas en nuestros papers, sus propuestas de mejora, su experiencia a la hora de elegir congresos y revistas donde publicar nuestros resultados, o su visión global cuando uno está demasiado centrado en lo técnico. Y al final, su comprensión cuando decidí cambiar de rumbo en lo profesional, su paciencia durante el tiempo en que no pude dedicar todo el tiempo que debía a la tesis, y su ayuda con la elaboración de este documento, han sido factores fundamentales en el resultado global de esta tesis. Por todo ello, ¡muchas gracias!

Durante estos años he tenido la suerte de compartir mis días con mis com-

pañeros de despacho, a los que tengo que agradecer todo lo que he aprendido con ellos, sin los que mi doctorado no hubiese sido igual. En primer lugar quiero acordarme de Íñigo. Juntos empezamos la tesis, en el mismo grupo, y vivimos las primeras alegrías y los primeros desengaños. Cuando llegamos los dos, allí estaban Alberto, Pablo, Fran, Carlitos, Juanan y Antonio (todos doctores hoy en día). De ese despacho han salido buenos amigos y no solo compañeros de trabajo, con visitas al Gallego cuando era menester, con muchos días de comer juntos, reir y preocuparnos juntos, y ... en esencia, vivir el día a día de la mejor forma posible. Tampoco quiero olvidarme de Guillermo, Poletti y Joaquín, con los que también he pasado momentos de todo tipo, y cuyas visiones, vivencias y consejos me han sido muy valiosos.

Entre todos aprendimos las reglas de la vida univesitaria (en el lado del profesorado), que en la Universidad nada es lo que parece, el quién es quién en este "Juego de Tronos", cómo funciona la investigación en general, y demás conjeturas. Inevitablemente, en cualquier aventura no todo lo que encuentras en el camino es siempre bueno, ni el camino es fácil, sobre todo en esta época de crisis. Por suerte, de mi experiencia universitaria me quiero quedar con lo mejor, y en ese saco están todos ellos.

Durante mi tesis he tenido la suerte pasar un total de 7 meses de mi vida (dividido en dos estancias) en Lyon, Francia, trabajando en el grupo de investigación AVALON perteneciente al INRIA. Meses que no olvidaré, en una magnífica ciudad que se hizo un hueco en mi corazón. Estoy muy agradecido a Frédéric Desprez, mi responsable allí, que gracias a su disposición en todo momento me hizo las estancias sencillas, como si estuviese en casa. En cuanto a mi trabajo, estoy tremendamente agradecido a Jonathan Rouzaud-Cornabas por el tiempo que trabajamos juntos y por su dedicación después de mis estancias. También quiero acordarme del resto del grupo de investigación, y especialmente de Daniel y Arrate, que me ayudaron a sentirme uno más dentro y fuera del despacho, y de George, con el que compartí meses de intenso trabajo en el despacho, y también muchos buenos momentos. Gracias a todos ellos tuve la oportunidad de estar en un ambiente de trabajo diferente al de Madrid, más enfocado a la investigación, y me llevo para el recuerdo una de las mejores épocas de mi tesis.

During this thesis, I had the opportunity of staying 7 months (divided into two periods) in Lyon, France, working within the AVALON research group, which belongs to INRIA. I will not forget these months in a gorgeous city which I carry in my heart. I am very grateful to Frédéric Desprez, my advisor there, who helped me a lot during both stays, taking care of whatever I needed. Regarding my work, I am really thanked to Jonathan Rouzaud-Cornabas for our work together during and after my stay there. I would like to thank to the rest of the research team, specially to Daniel and Arrate, who welcomed me to be part of their group inside and outside the lab, and to George, my office mate during the second period, with whom I spent a lot of time working and

laughing together. Thanks to all of them, I had the opportunity to be part of another work environment, more focused on research that the one in Madrid. These two stages are one of my best memories of this thesis.

En lo personal, quiero agradecer a mi familia el estar ahí cuando me ha hecho falta, en todos los sentidos. Con mis padres, en los momentos malos, he comentado 100 veces la necesidad de seguir adelante. Espero, en un futuro, darles la razón y decirles que tanto sufrimiento y que el tiempo empleado han merecido la pena. Mis hermanos nunca me han dicho otra cosa que no sean palabras de apoyo y de confianza en mi, cosa que es muy de agradecer. Se que esta tesis es también un triunfo vuestro, y quiero que sepáis que os lo merecéis, y que seais conscientes de que sin vuestro apoyo, esto no hubiese sido posible.

En especial quiero agradecer a Carolina el estar junto a mi en esta aventura, y soy consciente de que todo lo que diga es poco. Al empezar el doctorado en Madrid, me acompañaste, tuviste que buscar trabajo, que empezar una nueva vida. Durante estos años, he compartido contigo mis vivencias, la alegría de mis primeros artículos, los nervios con mis primeros viajes y charlas, lo bonito de conocer a mis nuevos compañeros, y . . . si, también verme llegar decaído muchas veces, las noches de deadline trabajando, y sobre todo la soledad de tenerme 7 meses fuera, o el verme salir del trabajo y emplear tantas y tantas noches y fines de semana trabajando en la tesis, en vez de hacer planes juntos. Por todo y por más, ¡muchas gracias!

Dans les champs de l'observation le hasard ne favorise que les esprits préparés.

Louis Pasteur

Siempre hay que saber cuándo una etapa llega a su fin. Cerrando ciclos, cerrando puertas, terminando capítulos; no importa el nombre que le demos, lo que importa es dejar en el pasado los momentos de la vida que ya se han acabado.

Paulo Coelho

Resumen

La tecnología Cloud Computing está cambiando la forma en que se proveen recursos, se despliegan y programan aplicaciones, y se entiende el mundo de las tecnologías de la información. Esta tecnología, basada principalmente en la virtualización y en el pago por uso, está en continua evolución y está siendo adoptada por muchas empresas privadas y públicas. Según ha ido desarrollándose la tecnología Cloud, un problema subyacente es la fragmentación del mercado de Cloud Computing en términos de ofertas de tipos de instancia, esquemas de precios y características que dan valor añadido a los recursos. Además, existe una gran dificultad a la hora de desplegar una infraestructura de forma óptima, cuando existen diferentes proveedores cloud disponibles. Por lo tanto, el objetivo de esta tesis es la investigación de mecanismos, técnicas y algoritmos para permitir un despliegue óptimo y un uso efectivo de aplicaciones multi-cloud, lo que se enmarca dentro del Cloud Brokering.

Las principales propuestas llevadas a cabo en esta tesis son las siguientes:

- Propuesta de una arquitectura de Cloud Brokering para el despliegue de infraestructuras virtuales en entornos multi-cloud estáticos y dinámicos. En los estáticos, ni los requisitos de usuario (ej, el número de cores), ni los parámetros del despliegue (ej, los precios) cambian a lo largo del tiempo. En los dinámicos pueden cambiar ambas cosas

- Propuesta de algoritmos de brokering destinados a la optimización de parámetros de la infraestructura a desplegar, considerando restricciones a dichos algoritmos que permiten un diseño más detallado del despliegue deseado. Estos algoritmos se enfocan en optimizar el coste total de la infraestructura (TIC) o el rendimiento total de la infraestructura (TIP). Las restricciones propuestas son: tipo de instancia, pudiendo elegir qué tipo de instancia usar; localización, pudiendo elegir qué proveedor cloud usar; y reubicación, pudiendo elegir qué porcentaje de infraestructura puede moverse de un cloud a otro.

- Propuesta y estudio de diferentes algoritmos de predicción de precios, dada la necesidad de predecir precios futuros de tipos de instancia en caso de utilizar precios dinámicos. El algoritmo propuesto está basa-

do en la media de los últimos datos de cada tipo de instancia, y la tendencia que estos datos llevan. Dentro del estudio comparativo, se proponen 3 algoritmos más de predicción de precios: last data, simple moving average y exponential moving average.

- Extensión de los algoritmos de planificación para tener en cuenta el almacenamiento de las instancias a la hora de desplegar de forma óptima la infraestructura.

Los resultados más remarcables son los siguientes:

- En experimentos sobre entornos estáticos, se demuestra cómo el bróker ayuda al despliegue óptimo de infraestructura entre diferentes proveedores cloud, sobre todo si el usuario conoce los requisitos exactos de la misma.
- En los primeros experimentos sobre entornos dinámicos, se demuestra que mover recursos entre diferentes clouds durante la vida de la infraestructura resulta más económico que dejándolos en un mismo cloud.
- En los experimentos considerando casos de uso reales (clusters genéricos, de clusters de altas prestaciones (HPC) y de servidores web), se demuestra:
  - Usar múltiples tipos de instancias mejora el resultado con respecto a usar un único tipo de instancias.
  - Cuando la carga de trabajo varía, el bróker ajusta la infraestructura necesaria automáticamente. Además, ajusta el número y tipo de instancias necesarios para lograr la optimización de coste.
- En los experimentos considerando almacenamiento, se demuestra que seleccionar las mejores políticas de almacenamiento, borrado y transferencia de imágenes puede reducir el coste del despliegue hasta un 90 %. La mejor combinación de políticas es almacenar la imagen en todos los clouds, no borrarla nunca y transferirla entre clouds en vez de hacerlo desde el bróker.

Como otras contribuciones, se destaca la aportación al simulador SimGrid Cloud Broker (SGCB), como parte de la colaboración pre-doctoral en el centro INRIA, Lyon.

Abstract

Cloud Computing technology is changing how computing resources are provided, applications are developed and deployed, and the way people understand the IT world. Cloud Computing is based on virtualization and pay per use, and it is continuously evolving and being adopted for private and public companies. While Cloud technology has been growing, an underlying issue is the fragmentation of the Cloud market in terms of instance types, pricing schemes, and value add features. Moreover, it is difficult to deploy an infrastructure in an optimal way, when different cloud providers are available. Therefore, the objective of this Phd. thesis is to research mechanisms, techniques, and algorithms to allow an optimal infrastructure deployment, and an effective use of multi-cloud applications. This objective is classified within Cloud Brokering topic.

The main proposals of this Phd. thesis are the following

- Proposal of a Cloud Brokering architecture for deploying virtual infrastructure in static and dynamic cloud environments. In static environments, neither user requirements (i.e. number of cores) nor deployment parameters (i.e. prices) vary along time. In dynamic environments, both parameters could change.

- Proposal of brokering algorithms to achieve optimal infrastructure deployments, considering constraints to allow a detail design of the desired deployment. These algorithms are focused on optimizing the Total Infrastructure Cost (TIC), or the Total Infrastructure Performance (TIP). The proposed constraints are the following: instance type, to choose which instance type to use; placement, to choose which cloud provider to use; and reallocation, to choose which percentage of infrastructure can be reallocated from one cloud provider to another.

- Proposal and research on different price forecasting algorithms, due to the necessity of predict future prices in dynamic environments. The proposed algorithm is based on each instance type last data average, and the trend of these data. Regarding the research comparison of forecasting algorithms, we consider three more algorithms: last data, simple moving average and exponential moving average.

- Consideration of instance type storage parameters within proposed scheduling algorithms.

As final remarks, these are the main conclusions:

- In the first static scheduling experiments, we demonstrate that users can find an optimal deployment if they know in advance the exact amount of hardware that they need.
- In the first dynamic scheduling experiments, we demonstrate that using the broker to move any percentage of the set of resources from one placement to another, always results in better choice than hold it in a static placement.
- In real use cases experimentation, we consider cases such as generic clusters, HPC clusters, and Web server applications. Here, we demonstrate the following:
  - Multiple instance type deployments out-perform single instance type ones.
  - In dynamic workloads, the broker select the best combination of instance types, reallocates the current infrastructure to reach the performance goal.
- In the storage experimental results, we highlight the significance of image transfer, deletion and storage policies for multi-clouds environments. We conclude that keeping images in every cloud during the deployment instead of uploading and deleting them when necessary, and using copy transfer strategy instead of get, can reduce TIC up to 90 %.

As other remarkable contributions, we highlight the improvement done to the simulator SimGrid Cloud Broker (SGCB), which was created for the experimental part of this thesis, thanks to a collaboration with AVALON research group, from Inria. Lyon.

Index

List of Acronyms

AMI Amazon Machine Image

AMPL A Mathematical Programming Language

API Application Program Interface

AS Asia (Amazon Region)

AWS Amazon Web Services

CaaS Computing as a Service

CIMI Cloud Infrastructure Management Interface

CMOS Configuration Memory Operating System

CPU Central Processing Unit

DaaS Desktop as a Service

DBaaS Database as a Service

DNS Domain Name Service

DMTF Distributed Management Task Force

E-A Everywhere-Always (storage policy)

E-N Everywhere-Never (storage policy)

EBS Elastic Block Storage

EC2 Elastic Compute Cloud

ECU Elastic Compute Unit

EMA Exponential Moving Average

EU Europe(Amazon Region)

FLOPS Floating Point Operations per Second

GCE Google Compute Engine

GCEU Google Compute Engine Unit

GFLOPS Giga(Billions) of Floating Point Operations Per Second

GPL General Public License

GPU Graphics Processing Unit

GUI Graphical User Interface

HA High Availability

HDD Hard Disk Drive

HPC High Performance Computing

IaaS Infrastructure as a Service

INRIA Institut National de Recherche en Informatique et en Automatique

IP Internet Protocol

IO Input/Output

IT Information Technology

LPD Last Period Data

NaaS Network as a Service

MFLOPS Millions of Floating Point Operations Per Second

MIT Massachusetts Institute of Technology

O-A OnDemand-Always (storage policy)

O-N OnDemand-Never (storage policy)

OCCI Open Cloud Computing Interface

ONE OpenNebula

OSI Open Systems Interconnection

PaaS Platform as a Service

PCI-DSS Payment Card Industry Data Security Standard

QoS Quality of Service

RAM Random-Access Memory

S3 Simple Storage Service

SaaS Software as a Service

SDN Software Defined Networking

SDK Software Development Kit

SGCB SimGrid Cloud Broker

SLA Service Level Agreement

SMA Simple Moving Average

SSD Solid State Drive

STaaS Data Storage as a Service

TIC Total Infrastructure Cost

TICS Total Infrastructure Cost with Storage

TIP Total Infrastructure Performance

USW United States West (Amazon Region)

VM Virtual Machine

VMI Virtual Machine Image

VPN Virtual Private Network

WMA Weighted Moving Average

Chapter 1

Introduction

If I have seen further,
it is by standing on the shoulder of
giants.
Isaac Newton.

In this initial Chapter we present and elaborate the motivation of this research, along with the objectives that we want to achieve in this work.

To understand our motivation and objectives, we introduce a general overview of the origins and evolution of provisioning models and platforms. We go from Mainframes to Clouds, digging into Clusters and Grids, giving a short definition of each technology, and comparing its features to its predecessors. The goal is to understand how Cloud computing, the technology involved in the development of this thesis, come into scene, its current state, and how it is evolving. We believe that a correct understanding of Cloud computing main principles is fundamental to follow this manuscript.

Within Cloud computing, we first introduce some definitions of the technology, and we also detail each of its main features: *virtualization*, *elasticity*, and the *pay-per-use* model. To finish with this overview, we discuss the adoption of Cloud computing by industry and private companies.

Then, we explain the motivation and objectives of this research. We mention the problem that we observe in Cloud computing nowadays, which is the fragmentation of the cloud market in terms of different features and offerings, and we elaborate on these differences to finish introducing our objective, which is to research on Cloud brokering mechanisms that allow an effective use of multi-cloud applications. As nowadays Cloud Brokering has several challenges, we also give a short overview of them, since we address them in this work.

Finally, we mention the contributions of this thesis, and we outline the organization of this work, introducing the main purpose of each chapter.

1.1. Evolution of provisioning models and platforms

The information technologies (IT) world has been experiencing a constant evolution in last 60 years, from in-house generated computing power to utility-supplied computing resources delivered over the Internet as Web services. Datacenters have evolved from expensive, rigid, mainframe-based architectures to agile distributed architectures based on commodity hardware that developers can dynamically shape, partition, and adapt to different business processes and variable service workloads [FZRL08].

Figure 1.1 depicts different periods from the beginnings of the computation to our era: large mainframes, isolated clusters, distributed grids, and clouds of virtual machines. The time-line represents the origin of these technologies and not the ending, since all of them are still in use.

The technology used in this thesis is the last one to appear, Cloud computing. However, a quick review of these technologies is essential to understand Cloud computing features.

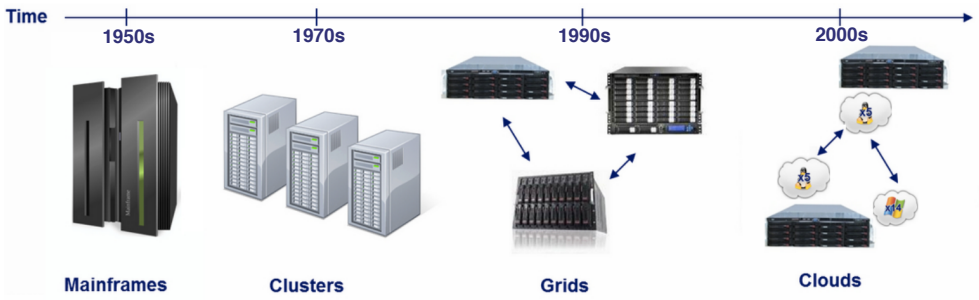


Figure 1.1: Distributed Computing Evolution

In the *Mainframe era*, one centralized and large system performed millions of operations to react appropriately to thousands of users requests. In its early years, microprocessors were slow compared to nowadays processors, and mainframes were physically large and very expensive. The limited and specialized use of them, made large computers a rare resource whose use was restricted to expert technologists and prestigious research laboratories. Generally speaking, the development of mechanisms for interacting with these computers remained restricted to computer scientists, mathematicians and engineers. Throughout most of the mainframe era, the power delivered to users had doubled approximately every 3.5 years, until the arrival of the complete processor on a CMOS chip, which conformed to Moore's law of doubling speed every 18 months.

The golden era of the mainframe thus eventually came to an inevitable end. This era collapsed with the advent of fast and inexpensive microprocessors, and data centers moved to collections of commodity servers. However,

mainframes are still alive both in industry - banks -, and research - modern mainframes still remain in top500 supercomputer list.

Cluster computing was born as the newest technology for computation:

"*A cluster is a type of parallel and distributed system, which consists of a collection of inter-connected stand-alone computers working together as a single integrated computing resource [Pfi98].*"

Similarly to mainframes, clusters tend to have high Input/Output (IO) throughput, same geographical location, high speed network between nodes, and a common set of applications installed. There are also some differences, such as clusters commodity networks versus mainframes specialized networks, as well as commodity versus specialized hardware, mainframes are homogeneous versus heterogeneous clusters, operating systems, or different programming models. Due to this technology, companies started to invest on his own clusters of machines instead of investing on mainframes.

Apart from its clear advantages, this new model inevitably led to isolation of workload into dedicated clusters, mainly due to incompatibilities between software stacks and operating systems. In addition, the unavailability of efficient computer networks meant that IT infrastructure should be hosted in proximity to where it would be consumed. This trend is similar to what occurred about a century ago when factories, which used to generate their own electric power, realized that it is was cheaper just plugging their machines into the newly formed electric power grid.

The next step in the computing evolution was the *Grid computing* technology, placed in 1990s. According to IBM [1],

"*a Grid is a type of parallel and distributed system that enables the sharing, selection, and aggregation of geographically distributed 'autonomous'resources dynamically at runtime depending on their availability, capability, performance, cost, and users'quality-of-service requirements.*"

Grid computing allows integrated, collaborative use of geographically separated, autonomous resources. As a result, systems scalability got highly improved compared to its predecessors, and some other known bottlenecks were avoided. One example of Grid-based technology is the Peer-to-Peer paradigm, which implies a direct communication between computers without centralized control, allowing a pair-based communication without intermediate host.

Ian Foster, one of the grid pioneers, define the main characteristics of a grid system as follow:

[1] IBM Solutions Grid for Business Partners: `http://joung.im.ntu.edu.tw/teaching/distributed_systems/documents/IBM_grid_wp.pdf`

> "*a system that coordinates resources, which are not subordinated to a central authority, and uses open, standard protocols and interfaces to provide not trivial qualities of services [FK99].*"

A grid serves a computational purpose and, analogously to the power grid, it focuses on demand-driven usage of standardized services with high reliability, high availability, and low-cost access. In contrast to cluster computing or mainframes, it does not require extremely high speed connections between nodes, allows geographical distribution, serves computations, and is theoretically application agnostic. Its innovation was not the power itself, but the coordinated distribution.

The *Cloud computing* technology emerged as the new paradigm of distributed computation after Grid computing, even it was explored for first time some years before its breakthrough. It just became more and more popular for the masses due to the improved network bandwidth offered in late 1990s by telecommunications companies, the growth of hypervisors usage, and the boom of the Internet. But, in fact, Cloud computing's roots (concept and characteristics) were thoroughly explored from 1950s-60s.

Cloud computing got his name from traditional network designs in which network engineers represented an unknown network using the symbol of a cloud. The rise of Internet has been the leading example of the cloud for the last many years. As a result, the cloud has come to represent the Internet, and adding computing to the equation makes Cloud computing become Internet Computing [2].

Cloud is on top of Grid and Clusters, and focusing on its features, Cloud provides more or less same functionalities as the other two, among others, but provides them in the form of services. Cloud combines features from grids, such as dynamical resource allocation, scalability or multi tenancy, and clusters, such as raw speed. Auto discovery and auto provisioning are also more native to the cloud, though grids have pioneered in that area. Some remarkable differences between clouds and its predecessors are the flexibility of having heterogeneous operating systems to build custom application environments, and the resource provisioning model, which uncouple software from hardware, and is done on-demand, via a web user interface or an API.

Clusters and grids are conceived for computing purposes, while clouds are more generic. In fact, clusters and grids can be run on top of clouds resources. From the computation point of view, clouds are more powerful than geographically distributed grids but less than centralized clusters, although performance is rather good to be close to some clusters performance.

After this brief explanation of Cloud computing we consider going deeper into this technology, so in next section we elaborate on this.

[2] Why it is called *Cloud Computing*?: `http://it.toolbox.com/blogs/original-thinking/why-is-it-called-quotcloud-computingquot-30713`

1.2. Cloud computing

Experts from different areas within IT define Cloud computing by trying to aggregate every feature or singularity about cloud systems in a single definition, within no more than 5 lines. As a result, in literature there are as many definitions as one could imagine. Cloud technology is continuously evolving, and its true value is making its presence throughout the entire IT ecosystem. It makes Cloud computing scope to get bigger and bigger, and therefore it is difficult to give a complete definition.

To illustrate some definitions, we choose two of them that are broadly accepted. R. Buyya, one of the highly cited authors in computer science and software engineering with more than 400 publications, define Cloud computing as follows.

> "*A Cloud is a type of parallel and distributed system consisting of a collection of interconnected and virtualized computers that are dynamically provisioned and presented as one or more unified computing resources based on service-level agreements established through negotiation between the service provider and consumers (R.Buyya et al. [BYV08]).*"

And Vaquero et al. in [VRMCL09] propose a definition which is made from the main available definitions extracted from literature, to provide both an integrative and an essential Cloud computing definition. It is the following:

> "*Clouds are a large pool of easily usable and accessible virtualized resources (such as hardware, development platforms and/or services). These resources can be dynamically reconfigured to adjust to a variable load (scale), allowing also for an optimum resource utilization. This pool of resources is typically exploited by a pay- per-use model in which guarantees are offered by the Infrastructure Provider by means of customized SLAs.*" [VRMCL09]

Looking forward into these definitions, it is important to emphasize the key words that make Cloud computing different from previous technologies, and give a real overview of the foundations on which it is based:

- *Virtualization.*
  Virtualization is the key enabler technology of clouds, as it is the basis for most of the features, such as on demand sharing of resources, resource elasticity, or resource scalability, among others.

  Cloud computing services are usually backed by large-scale data centers composed of thousands of computers. Such data centers are built to serve many users, and host many disparate applications. For this purpose, hardware virtualization can be considered as a perfect fit to overcome most operational issues of data centers, and it also relieves

most of the issues that have caused frustration when using grids, such as hosting many dissimilar software applications on a single physical platform.

Within Cloud computing, computation, storage, and network resources are virtualized, and offered as a service. This makes virtual machines images portable across sites, and provides resource isolation in a secure way, among others. Consequently, these virtualized infrastructures are considered a key component to drive the Cloud computing paradigm.

Apart from that, there are lots of features that distinguish Cloud computing from other technologies. Some of them are shared with Grid computing, some others with Cluster computing, and a few of them that are completely new.

- *On-demand provisioning.* Cloud computing allows to use resources as they are needed. This feature is disruptive if we compare it with clusters, for example, where getting new resources takes days or weeks. Therefore, with Cloud computing the resource provisioning should be done on-demand, in the sense that it should be done appropriately and quickly when a resource is needed.

- *Elasticity.* One of the key benefits of using Cloud computing is the elasticity that it provides to users services when required. This is possible by increasing the number of virtual resources, or by the automatic resizing of them on an on-demand basis. Most cloud service providers allow users to increase their existing resources to accommodate increased business needs or changes. Therefore, elasticity requires dynamic reconfiguration in most cases, and as the system scales, it needs to be reconfigured in an automated manner. This feature allows users to support their business growth without expensive changes to their existing IT systems. As an example, if on a particular day the user's demand increases several times, the system should be elastic enough to meet that additional need, and it should return to a normal level when this demand decreases.

- *Multi tenancy.* In Cloud computing, virtual resources are pooled to serve a large number of customers. It implies multi-tenancy, where different virtual resources are dynamically allocated and de-allocated according to demand, and different tenants'resources share the same physical infrastructure. From the users end, it is not possible to know exactly where the resource actually resides.

- *Service oriented.* Cloud computing transforms resources and business processes into software services, and expose them using APIs (Application Programming Interfaces). Some examples, among others, are

Infrastructure as a Service (IaaS), where users can request virtual machines, storage, or virtual private networks, or Database as a Service (DBaaS), where users can execute their queries against their database engine, bot via APIs, against a determined endpoint url.

The IaaS or DBaaS underlying architecture is abstracted without exposing much to user, through the use of virtualization and other technologies. Abstraction and accessibility are two keys to achieve the service oriented conception [GLZ+10]. The service-oriented approach helps businesses adapt to change, and makes the scalability of Cloud computing possible.

- *Pay-per-use model.* Cloud computing introduces the pay-as-you-go billing model, which means that users only pay for the amount of resources that they use. For example, using some hours a certain virtual machine, uploading several gigabytes of storage to some storage server, or executing a thousand queries against some database.

  Cloud resources need to be available at any time, and depending on users demand, cloud providers scale up or down its resources dynamically. Therefore, providers can reproduce its internal resource consumption to user bills.

  About pricing, as the user is billed based on the amount of resources they use, cloud providers offer clients means to capture, monitor, and control usage information for accurate billing, and bring it to them in a transparent and readily way.

- *Service Level Agreements (SLA).* As cloud users move towards adopting such a service oriented architecture, the quality and reliability of the services become important aspects. Users demand for cloud services vary significantly over time, and it is not possible to fulfil all users quality expectations from the service provider perspective. Therefore, the figure of an SLA appears [PP09], where cloud providers and users commit to an agreement via a negotiation process. Generally, these SLAs assure users certain minimum resource deployment capacity, protection against providers physical infrastructure failures, as well as high cloud services availability.

In Chapter 2 we cover in depth the aforementioned Cloud computing features, and also the Cloud computing service and deployment models. Here we introduce a global view of these models:

- There are three main cloud service models: Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS).

  IaaS refers to the deployment of virtual machines, also considering storage or network; PaaS delivers a development platform to supports

the software life-cycle; and SaaS refers to the use of software that run on top of cloud infrastructure.

- Within IaaS, there are three cloud deployment models: Private, Public, and Hybrid.

  Private clouds are developed, operated, and managed within a single organization; Public clouds are developed and managed by third companies, which offer access to external users at a certain cost; and Hybrid clouds are a combination of infrastructure deployed in two or more clouds, mainly private and public.

Cloud public adoption

One main difference between Cloud computing and previous technologies is the number of private institutions that invest on adopting this technology. In previous technologies, developments were supported mainly by academia and public institutions, and partially from private ones. However, Cloud computing development is supported in high percentage by big private companies that focus their business on this technology, and want to adopt it as soon as possible [DM14] [DMCE15].

As Cloud computing gained adepts in private business, several milestones became. One of the first milestones for Cloud computing is the arrival of Salesforce.com in 1999, which pioneered the concept of delivering enterprise applications via a simple website. The next milestone is the arrival Amazon Web Services in 2002, which provided a suite of cloud-based services including storage and computation. Then in 2006, Amazon launched its Elastic Compute Cloud (EC2) as a commercial web service that allows small companies and individuals to rent computers on which to run their own computer application. Another big milestone came in 2009, as Web 2.0 hit its stride, and Google and others started to offer browser-based enterprise applications through services as Google App. And also in 2009 the industry saw the advent of Microsoft into the Cloud computing game with its product Windows Azure.

And now, many IT professionals recognize the benefits Cloud computing offers in terms of unlimited resources, flexibility and cost reduction. Other considerations, such as security, data privacy, network performance, and economics, are likely lead to a mix of private and public Cloud computing based data centers, all controlled by an institution.

1.3. Motivation and objectives of this research

Cloud computing technology, contrarily to previous technologies, is being widely adopted by many private companies, some of them acting as consu-

mers (i.e, purchasing virtual machines to face companies commitments with their clients); and some of them acting as technology providers (i.e, acquiring physical infrastructure, the necessary human resources to maintain it, and then exposing its services to everyone using the Internet).

In the consumer side, many cloud experts agree on the benefits that Cloud computing can give to these companies, from big ones to start-ups. One of the main advantages is the pay per use, instead of investing on physical machines and its maintenance.

Regarding providers, we observe that the number of providers in the Cloud computing market increase at rapid pace as the technology is being adopted. For example, Amazon is the pioneer provider of Cloud Computing resources with its Elastic Compute Cloud (EC2). Other providers joined Amazon in the cloud market, such as Google with its Google Compute Engine (GCE) [3], or Microsoft with Azure [4].

In terms of funding and reputation, these are some of the most famous ones, but nowadays companies do not need to be as big as the aforementioned companies to became a cloud provider. Some examples are ElasticHosts [5], GoGrid [6], Flexiant [7], or Rackspace [8], among a large number of companies that purchased their private infrastructure, set its private cloud up by installing hypervisors and cloud management platforms on top of their infrastructure, and created a Web interface to sell and manage their infrastructure and services. As different providers appeared, the market offerings became complex for users, and cloud provider comparisons became also difficult.

PREMISE: As a result, the problem we observe is the fragmentation of the market in terms of virtual machine offers, pricing schemes, and value-add features, and the difficulties to deploy and manage a virtual infrastructure in an optimal way, when multiple providers are available.

The cloud offer is highly heterogeneous, where each provider offers what they consider appropriate. Here we expose some examples:

- Virtual machine offers: initially only three types of instance were offered by Amazon: small, large and extra large, with its particular fixed resources (CPU, RAM memory, and disk storage). Then, other providers offered different composition of resources with the same name of instance type. And later, other providers offered different instance types (i.e. different compositions and different name), or different value-add

[3]Google Compute Engine - `https://cloud.google.com/products/compute-engine/`
[4]Windows Azure - `http://www.microsoft.com/windowsazure/`
[5]ElasticHosts - `http://www.elastichosts.com/`
[6]GoGrid - `http://www.gogrid.com/`
[7]Flexiant - `http://www.flexiant.com/`
[8]Rackspace - `http://www.rackspace.com/`

features, such as increasing the amount of CPUs without restarting the virtual machine, or instances with GPUs.

- Pricing schemes: in the early phase of cloud adoption the price model was dominated by fixed prices. Similarly to the instance types, each provider set different prices for the VM offers. Moreover, some providers adopted different pricing periods (from hourly to five-minutes rates), and different pricing discounts, if users lease resources in a long term manner (i.e. 3 months, 6 months or 1 year).

  Later, the use of dynamic pricing schemes increased. Dynamic prices were only offered by Amazon, and nowadays it remains the same. In the dynamic plan, prices change according to demand, so users can take advantage of idle time in certain cloud provider resources by purchasing the instance type they need at a lower price. The disadvantage of current dynamic pricing scheme is that, when demand raises, the provider can shut-down the virtual machine without asking the user. This is a risk that the user may take in exchange for the reduced price.

- Value-add features: in parallel to the evolution of virtual machines and pricing schemes, cloud providers increased the number of features offered together with VMs, such as firewalls, load balancers, public IP addresses, virtual private clouds, or monitoring systems, among others.

With all of these different conditions, it is difficult for cloud users to search and decide where to deploy their resources. Moreover, nowadays it is difficult to achieve the following goals during an infrastructure deployment:

- Optimize VM distribution among available cloud providers and instance types.
- Optimize costs deploying infrastructure in the best available providers.
- Improve deployments to became high available, deploy geographically distributed infrastructure, or avoid vendor lock-in.

So, our motivation is to create a tool that could help everyone to deal with these changing conditions, and to achieve the aforementioned goals.

And therefore, our objective in this work is to research mechanisms, techniques, and algorithms to allow an effective use of multi-cloud applications.

Cloud brokering is the topic that we consider to research, and within Cloud brokering, we address the following challenges in this work:

- Brokering scenarios: deploy infrastructure in static and dynamic scenarios in an autonomous way.

  We define static and dynamic scenarios, as cloud environments in which deployment conditions do not change (static) or can change (dynamic) along the infrastructure lifetime. About these deployment conditions, we consider user requirements, such as number of required virtual machines, which can be dynamic depending on the moment of the day, or cloud features, such as resource prices, or resource availability, among others.

  Several research works in the field [TMMVL12] study how to take advantage of Cloud brokering features under static conditions, e.g. when provider and user conditions do not change. These works reveal optimal deployments in several use cases, scheduling the virtual infrastructure once and deploying it in the best providers. However, when the virtual infrastructure life-time is long enough, cloud provider conditions can change (e.g. prices), so it is necessary to analyse how to optimally reconfigure the service to adapt it to new situations.

  In dynamic scenarios, e.g. if a new cloud provider appears [9], an instance type is retreated/added from/to the cloud market [10] [11], the user needs change, or prices change along the infrastructure life time, it is possible to obtain a better placement of the resources by reallocating the current infrastructure to some different clouds. For instance, pricing schemes can differ by vendor, or even prices can vary dynamically based on current demand and supply (e.g. Amazon EC2 spot prices). These differences provide users the chance to compare providers and reduce their virtual infrastructure investment [YKA10].

- Scheduling strategies: optimize deployment parameters, such as cost or performance, indicating certain constraints to meet in each deployment. As an example, cloud users can control their deployment investment by setting cost limits, assure a certain deployment performance during each moment of their infrastructure lifetime, or use their favourite clouds to deploy it.

  We focus our research in optimizing the total infrastructure cost, or the total infrastructure performance. We define both of them in mathematical notation as our objective functions, and we can not use both at the same time.

[9] Announcing the AWS Asia Pacific Singapore Region. 2010 - `http://aws.amazon.com/about-aws/whats-new/2010/04/29/announcing-asia-pacific-singapore-region/`

[10] Announcing Micro Instances. 2010 - `http://aws.amazon.com/es/about-aws/whats-new/2010/09/09/announcing-micro-instances-for-amazon-ec2/`

[11] Announcing Cluster GPU instances. 2014 - `http://aws.amazon.com/es/about-aws/whats-new/2010/11/15/announcing-cluster-gpu-instances-for-amazon-ec2/`

Moreover, we define certain constraints to these objective functions in order to make the cloud broker more flexible for everyone. Therefore, users can adjust their deployments to their needs, and do it in a strict way. Some of these constraints are placement constraint, to choose in which clouds we want to deploy the infrastructure, or instance type constraint, to choose which instance types to use.

- Pricing schemes: as prices change, we need to forecast forthcoming prices to deploy optimally the required infrastructure.

  We consider current cloud market pricing schemes, such as on-demand or spot schemes, among others. Here the challenges are to select the best pricing scheme considering infrastructure life time, and to estimate forthcoming prices to deploy the infrastructure in the cheapest cloud in a dynamic scenario.

- Finally, the challenge of creating a cloud broker to include our proposals and to help us with the experiments of our research.

In the experimental section we test our proposal for each aforementioned challenge. We perform experiments with statics and dynamic scenarios, optimizing infrastructure cost and performance, and using every constraint that we propose. In short, we considered the following stages:.

- In a preliminary stage, our focus is to develop and set up our simulator with non-complex scenarios.

- Considering complex scenarios, our focus is to experiment with real world use cases deployments, such as generic clusters, HPC cluster, and Web Servers.

- In the final stage, our focus is to introduce storage parameters in the process of Cloud brokering.

In this thesis we make the following contributions:

- Proposal of a novel cloud broker architecture adapted to multi-cloud environments, designed to be aware of different cloud features, that acts as a cloud management software. One of the main components of the broker architecture is the cloud scheduler, which is responsible for making autonomously scheduling decisions based on scheduling strategies.

- Research on different scheduling strategies to include and configure in the scheduler, based on different optimization criteria, such as service cost, service performance, or custom implemented ones. According to these policies, the scheduler performs an optimal deployment of the service components among different cloud providers trying to optimize the selected criterion.

- Implementation of these strategies into SimGrid Cloud Broker (SGCB) simulator. SGCB was created during this thesis in collaboration with INRIA research group (Lyon, France). Currently, we are contributors of SGCB, which is the tool we considered to perform part of our experiments.

To finish this Chapter, we depict how this thesis is organized:

- Chapter 2 reviews the state of the art from traditional mainframes to current cloud infrastructures, describing deeply how clouds have evolved to federations of clouds, and the role of cloud brokering within them.
- Chapter 3 presents the main contribution of this thesis: the architecture for Cloud brokering, the scheduling strategies proposed, and the improvements introduced along this work.
- Chapter 4 describes the experimental results of applying the brokering strategies to some real world industry-relevant use cases.
- Our concluding remarks will be given in chapter 5, as well as the future lines of work.

Chapter 2

Cloud computing technology

Si no conozco una cosa, la investigaré.

Louis Pasteur

In this Chapter we explain the state of the art on Cloud computing, going from general concepts of this technology, to concrete concepts of the main topic of this thesis, Cloud brokering.

First, we introduce Cloud computing architecture and service models, explaining in detail the architecture layers and the most popular service models: Infrastructure as a Service, Platform as a Service, and Software as a Service. In this work, we focus on Infrastructure as a Service (IaaS) model.

Then, we describe the taxonomy of IaaS clouds, the cloud deployment models, and their features. These deployment models are Private, Public, and Hybrid clouds. In this work, we consider multi-cloud environments mainly composed by public clouds.

We introduce the concept of Cloud federation among different cloud providers. The goal of Cloud federation is to enable cloud providers and IT companies to collaborate and share their resources to fulfil requests during peak demands, and negotiate the use of idle resources with other peers. We study Cloud federation goals, challenges, and types of architectures. After that, we present Cloud brokering and its current state of the art, mentioning current public and private efforts in the field.

Finally, we introduce a technical section to explain the adoption of Cloud brokering technology for industry. Therefore, this section focuses on private companies, and here we expose techniques to build a private cloud, and libraries and tools to interact with clouds.

2.1. Architecture and service models.

The *architecture* of a Cloud computing environment can be divided into 4 layers [ZCB10]: the hardware/datacenter layer, the infrastructure layer, the platform layer and the application layer, as shown in Figure 2.1.

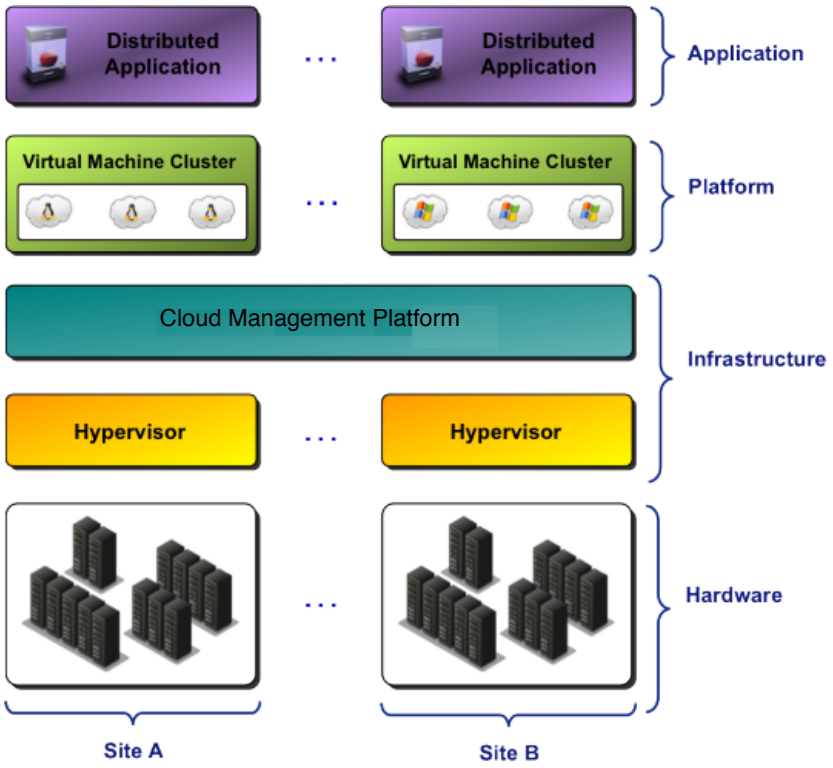


Figure 2.1: Cloud Computing Architecture

The *hardware layer* is responsible for providing the physical resources of the cloud, including physical servers, routers, switches, power and cooling systems. In practice, the hardware layer is implemented in data centers. A data center normally contains thousands of servers organized in racks, and interconnected through switches and routers. Typical issues at hardware layer include hardware configuration, fault-tolerance, network traffic management, or power and cooling resource management.

The *infrastructure layer* allows the creation of a pool of storage, network, or computing resources by partitioning physical resources using virtualization techniques. Considering computing virtualization, hypervisors allow the creation and execution of multiple virtual machines on the same physical machine. Some examples of hypervisors are Xen [B+03], KVM [KKL+], and VM-

ware [1]. Cloud management platforms allow the management of the life-cycle of these virtual machines. Some examples of cloud management platforms are OpenNebula [2], and OpenStack [3]. Considering network virtualization, Software Defined Networking (SDN) emerged as a paradigm that makes it easier to create virtual networks and introduce new abstractions in networking, simplifying network management and facilitating network evolution. SDN separates the network's control logic from the underlying routers and switches, promoting the centralization of network control, and introducing the ability to program the network [KREV+15] [NMN+14]. Some examples of SDN tools are OpenDayLight [4], and Open vSwitch [5]. The infrastructure layer is essential in Cloud computing, since the use of virtualization and cloud management platforms is fundamental to make possible some of the key features of this technology, such as better utilization of physical resources, dynamic resource assignment, resource scalability, or management and monitoring of virtual machines.

The *platform layer* consists on groups of virtual machines with embedded software (operating systems and application frameworks), that act as middleware for a running application. The purpose of the platform layer is to minimize the burden of deploying applications directly into virtual machines. For example, Google App Engine operates at the platform layer to provide Google users with API support for implementing the storage, database, or business logic of, for example, typical web frameworks or modern Hadoop frameworks.

At the highest level of the hierarchy, the *application layer* comprehend the current cloud applications, such as specific software designed to run on multiple and possibly distributed machines, and also legacy applications, which are not designed for cloud environments but which also can be run on top of them. Both type of applications can leverage the automatic-scaling feature to achieve better performance, availability and lower operating cost.

Compared to traditional service hosting environments such as dedicated server farms, the architecture of Cloud computing is more modular. Each layer is loosely coupled with the layers above and below, allowing each layer to evolve separately. This is similar to the design of the OSI model for network protocols. The architectural modularity allows Cloud computing to support a wide range of application requirements while reducing management and maintenance overhead.

Considering *service models*, there are three well established service models within Cloud computing: Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS) [KF08], which are depicted

[1] VMware Virtualization Software for a Private Cloud - `http://www.vmware.com/`
[2] OpenNebula home page - `http://www.opennebula.org/`
[3] Open Stack Open Source Cloud Computing Software- `http://www.openstack.org/`
[4] Open Daylight - `http://www.opendaylight.org/`
[5] Open vSwitch - `http://openvswitch.org/`

in Figure 2.2, and explained below.

However, through the cloud everything can be be delivered as a service, from computing power to business processes to personal interactions. For example, *Data storage as a Service (STaaS)* [HIM02], in which virtualized storage on demand becomes a separate cloud service (e.g. Amazon S3, Google BigTable, Apache HBase), *Computing as a Service (CaaS)*, *Desktop as a Service (DaaS)* [KKU12], or Network as a Service (NaaS) [ZZZQ10].

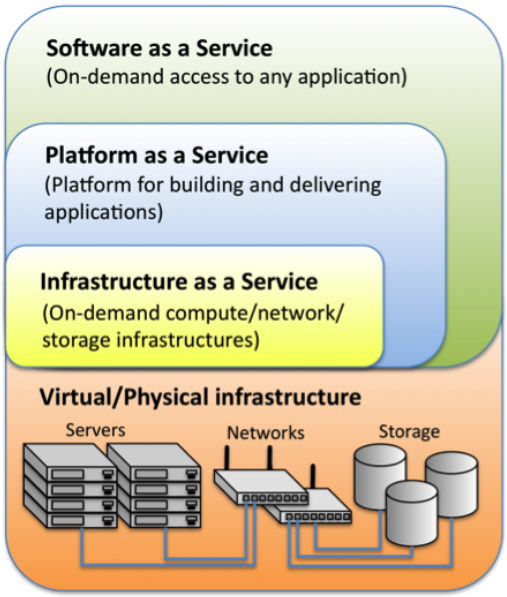


Figure 2.2: Cloud Computing Service Models

2.1.1. Infrastructure as a service.

Infrastructure as a service (IaaS) is the model that manages the Hardware and Infrastructure layers of Figure 2.1. It is a standardized, highly automated model, where compute resources, storage, and networking capabilities are owned and hosted by a cloud provider, and offered to customers on-demand.

Customers are able to self-provision their infrastructure using a Web-based user interface, which serves as an operations management console for the overall environment. Moreover, cloud providers offer an API access to the infrastructure, which we consider a disruptive feature.

Virtualization is extensively used in IaaS model in order to meet growing or decreasing resource demand, by creating or terminating virtual machines within physical resources. The strategy of using virtualization is to set up independent virtual machines, isolated from the underlying hardware, and also isolated from other VMs. Some examples of public IaaS providers are

Amazon EC2 [6], RackSpace [7], or ElasticHosts [8].

2.1.2. Platform as a service

Platform as a service (PaaS) is the model tied to the Platform layer of Figure 2.1. Platform as a Service delivers a development platform that supports the full software life-cycle, which allows cloud consumers to develop cloud services and applications directly on the cloud. PaaS is designed to host production services as well as applications being developed. Moreover, it offers the possibility of update these applications, and assign resources on-demand regarding application load. In addition, PaaS allows users to own their development infrastructure, including development environments, tools, or a centralized configuration management [DWC10].

An example of PaaS is Windows Azure [9] services with the web role, worker role, or reporting services. Another example of PaaS is *Google App Engine* [10], in which Google users can build their applications on top of Google-managed data centers, taking advantage of advanced features in a transparent way, such as automatic scaling for web applications; or *Heroku* [11], which has support for many programming languages such as Java, Node.js, Scala, Clojure, or Python.

Moreover, there are some open source PaaS projects that anyone can install on his/her premises like OpenShift [12] (RedHat PaaS solution), or Stratos [13] (Apache OpenSource project). These solutions are not as complete as the previous ones, but they enable medium/small companies a way to offer their developers a private and controlled environment to build their applications.

2.1.3. Software as a service

Software as a service (SaaS) is the model focused on the Application layer of Figure 2.1. Software as a service can be defined as software that is owned, delivered, and managed remotely by one or more providers. The provider delivers software based on one set of common code and data definitions, that is consumed in a one-to-many model by all contracted customers, at any time, on a pay-for-use or as a subscription basis. Applications can be accessed through the Internet from different clients (e.g, web browser, or

[6] Amazon Elastic Compute Cloud (EC2) - `http://aws.amazon.com/ec2/`
[7] Rackspace - `http://www.rackspace.com/`
[8] ElasticHosts - `http://www.elastichosts.com/`
[9] Windows Azure - `http://www.microsoft.com/windowsazure/`
[10] Google App Engine - `http://code.google.com/intl/es-ES/appengine/`
[11] Heroku home page - `https://www.heroku.com/home/`
[12] Open Shift by RedHat - `https://www.openshift.com/`
[13] Apache Stratos - `http://stratos.incubator.apache.org/`

smart phones) of application users, which do not have control over the cloud infrastructure.

Examples of SaaS include SalesForce.com, Google Mail, or Google Docs. In case of Gmail, it is a type of a SaaS mail provider since users do not have to manage any service themselves, and it is the vendor who takes care of it.

2.2. Taxonomy of IaaS clouds

In this work we focus on the Infrastructure as a Service service model, and in this section we explain the three different IaaS deployment models: private, public, and hybrid (see Figure 2.3).

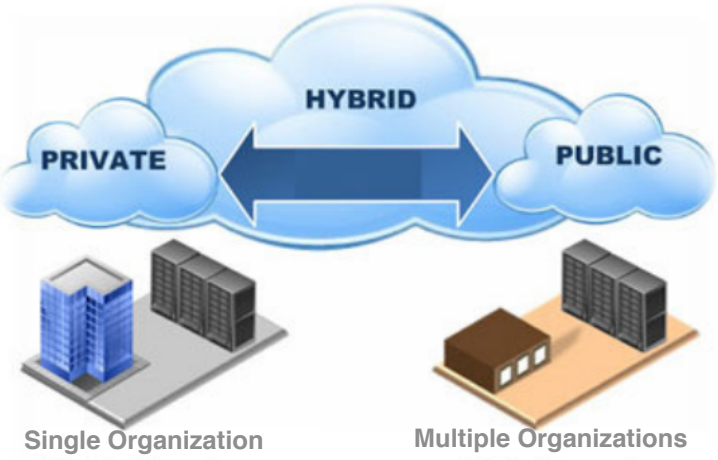


Figure 2.3: Cloud computing deployment model.

2.2.1. Private clouds.

The cloud infrastructure is operated solely within a single organization, and managed by the organization or a third party regardless whether it is located on premise or off premise [DWC10]. Private clouds are internally managed using cloud management platforms, such as the aforementioned OpenNebula, or OpenStack.

Most organisations consider private clouds to be the most secure and efficient way of Cloud computing. The primary goal of private clouds is not to sell capacity over the Internet through publicly accessible interfaces, but to give organization users a flexible and agile private infrastructure to run service workloads within their administrative domains [SMLF09].

The main advantage of private clouds is that physical resources are fully dedicated to users or projects within the organization, contrarily to public

clouds, where physical resources are shared with other people. Hence, organization users with administrative privileges get all the control of the server, so they can secure it through custom firewalls, or make backups and store data on their machines.

However, private clouds present drawbacks mainly on infrastructure acquisition and maintenance costs, and also on creation and management of virtual machine images, service appliances, or security concerns, among others. Generally, it costs more than other deployments, not only for the physical resources, but also because good administrators are essential to configure all aspects of the cloud, and customize firewalls or certain software to protect it. Private clouds can also support a hybrid cloud model, by supplementing local infrastructure with computing capacity from an external public cloud.

2.2.2. Public clouds.

Public clouds are the dominant form of current Cloud computing deployment model. Public clouds appeared when commercial cloud providers offered a publicly accessible remote interface for creating and managing VM instances, within their proprietary infrastructure, at a certain cost [SMLF09]. Public clouds offer access to external users who are typically billed on a pay-as-you-use basis.

The advantages presented by public clouds are focused on resource availability, since users can purchase almost unlimited virtual resources within minutes, cost saving, since users only pay for what they use and avoid taking care of machines maintenance, among other costs, and easy to use, since providers offer several ways to interact with the infrastructure, such as graphical user interface (GUI), APIs, or proprietary tools, such as *ec2-tools* from Amazon, or *gcutil* from Google.

Public clouds are generally easy to administrate, and there are automated software for daily operations available in different communities. However, they present drawbacks like resources sharing, since virtual machines that belong to users from different organizations share physical resources. Hence, there is not complete security for crucial data since these resources can not be entirely dedicated to single organizations.

Public cloud providers

Nowadays there are many public cloud providers, companies that invest money on purchasing infrastructure to sell its services to their clients. The main competitors in this world are mainly three: Amazon EC2 [14], Google Compute Engine [15], and Microsoft Azure [16]. However, in this work we con-

[14] Amazon Elastic Compute Cloud (EC2) - `http://aws.amazon.com/ec2/`
[15] Google Compute Engine - `https://cloud.google.com/products/compute-engine/`
[16] Windows Azure - `http://www.microsoft.com/windowsazure/`

sider also some upstart companies, such as OpSource [17] among others.

At the beginning of this work we consider to analyse the most important providers regarding certain criteria, such as resource offers (compute, memory, and storage), security, content and marketplace, or pricing schemes. Here we depict our evaluation, considering only the main cloud providers:

- *Amazon EC2* is the more complete cloud provider in cloud market considering the aforementioned criteria. It is also the oldest one, with more than seven years in the market.

  Amazon offers the highest number of instance types, pricing schemes, locations, or value-add services around its infrastructure. Moreover, its Spot pricing scheme is unique in the market. And its marketplace is really complete, since it has lots of base images, third parties images, or community images with possibility of sharing them between users.

  Considering security, Amazon holds several security certifications, such as PCI-DSS among others. Generally, most of cloud providers hold approximately certain level of security certifications.

- *Google Compute Engine* is currently behind Amazon EC2. Although it is one of the biggest cloud providers, it is inside the market from no more than 3 years.

  Google Compute Engine offers standard pricing schemes, but generally its prices are cheaper than their competitors. It also offers less types of instance that Amazon, and its market is not as complete as Amazon, mainly on base images to deploy, and third parties customized images. Regarding security and reliability, it is close to Amazon.

  They hold some disruptive features, such as the billing period (each five minutes), or their solution for networking, which is based on Software Defined Networking (SDN), one of the most promising upcoming technologies.

- *Microsoft Azure*, with its VM role, is far away from the aforementioned competitors. Although they are not new in the market, they envisioned Cloud computing more from a Platform as a Service point of view than their competitors. Therefore, their Infrastructure as a Service offering is not as significant as other providers.

  Their advantages are the performance of Microsoft Windows virtual machines, and the value-add proprietary services that they offer, such as the Office suite, among others. However, prices in general are more expensive in Azure than its competitors.

[17] Op Source, a Dimension Data Company - `http://www.opsource.net/`

- *OpSource* is not as big as their competitors, even they arrived in the market before Google Compute Engine.

  The remarkable advantage between OpSource and the others, is that users can upload its own virtual machine image to their cloud, and also that their clients can customize the instance type that they want, instead of using only pre-defined ones.

Instance Types

Cloud providers offer a wide range of instance types taking into account physical features such as CPU, RAM memory, or hard drive capacity. They classify these instance types into pre-defined and customizable instances:

- Pre-defined instances are those which can not be modified by users. Each provider offers its own range of pre-defined instances, but some are common to the majority of them, such as the Small (S), Medium (M), Large (L), and Extra Large (XL) instance types, since these instance types are the basic ones.

  Currently, there are lots of pre-defined types that take into account user needs, such as the memory optimized *r2-family*, or the compute optimized *c3-family* from Amazon EC2. There are also instance type families that optimize storage and GPU compute within EC2.

  As an example, Table 2.1 shows some pre-defined instance types from Amazon EC2 and Google Compute Engine.

Amazon EC2				
	Standard (m1)			
Inst. type	**small**	**medium**	**large**	**xlarge**
CPU (ECU)	1	2	4	8
RAM (Gb)	1,7	3,75	7,5	15
Storage (Gb)	160	410	2x420	4x420
Google Compute Engine				
	Standard			
Inst. type	**n1-std-1**	**n1-std-2**	**n1-std-4**	**n1-std-8**
Cores (GCEU)	1	2	4	8
RAM (Gb)	3,75	7,5	15	30
Storage (Gb)	0	0	0	0

Table 2.1: Instance types features.

It is important to note the heterogeneity of CPU units among public cloud providers. Amazon defined Elastic Compute Units (ECUs) as an

abstraction of computer resources. One ECU provides the equivalent CPU capacity of a 1.0-1.2 GHz 2007 Opteron or 2007 Xeon processor. However, Google defined GCEU (Google Compute Engine Unit) for the same purpose. One GCEU provides the equivalent CPU capacity of 2.75 GQ's on Sandy Bridge processor.

- Customizable instances allow users to make their own instance by choosing the amount of resources they need. Therefore, users can have virtual machines explicitly defined for its applications (e.g. 4 CPU cores with 1Gb of RAM for compute intense applications, or 1 CPU core with 8Gb of RAM if it is needed). Also users can benefit from it because they only pay for what they need, not for excessive and infra-utilized resources.

Pricing Schemes

Pricing of the cloud resources is both a fundamental component of the cloud economy and a crucial system parameter for the cloud operator, because it directly impacts customer. From the beginning of the cloud providers, two strategies have been used for pricing: static and dynamic pricing.

Static pricing remains the dominant form of pricing today. Within static pricing model, there are two schemes that are well established in almost every provider:

- On-demand pricing scheme allows users to pay for what they use, charging them for a certain minimum periods (such as 5 minutes, or 1 hour) depending on the provider. During this period, users can stop/start the virtual machine every time they need.
- Long reservation pricing scheme allows users to save money by reserving resources for long periods of time (e.g, one month, one year, or three years). Providers offer discounts that can benefit users which know exactly their applications lifetime.

However, dynamic pricing model emerges as an attractive strategy to better utilize unused cloud capacity, while generating an extra revenue to cloud providers. Spot pricing scheme is the best example of this policy. The spot pricing scheme was made according to the characteristics of Amazon EC2's spot instances [YKA10]. Amazon provides a spot instance when a user's bid is greater than the current price, and stops immediately without any notice when a user's bid is less than or equal to the current price. Amazon does not charge the latest partial hour when Amazon stops an instance, but it charges the last partial hour when a user terminates an instance. The price of a partial hour is considered the same as a full hour, and Amazon charges each hour by the last price. Finally, Amazon freely provides the spot price history.

Value Added

Cloud providers include in their offerings different value-add services or features, generally without an extra cost. Some examples of services are load balancers, tools to create subnets, or firewalls, among others networking services; or the resource consumption tools that some public cloud providers offer to users, allowing them to monitor and control their investment in cloud services in real time, and to use alerts or set consumption limits. There are other services, such as monitoring tools, in which the basic version is offered for free, but it is possible to use an extended version if the user assumes its cost.

As value-add features, providers generally hold certain certifications like ISO 27001, which is focused on security, or PCI DSS (Payment Card Industry Data Security Standard), which is focused on the payment industry. Moreover, providers assure users a certain SLA (Service Level Agreements) for their virtual infrastructure, which is usually very high. This guarantees users that their infrastructure will not fail, generally between 95 % and 99 % of its lifetime.

2.2.3. Hybrid clouds.

The hybrid cloud model refers to a combination of infrastructure deployed in two or more different clouds, private and public clouds. These clouds remain as unique entities, but are bound together by standardized or proprietary technology, that enables data and application portability (e.g, cloud bursting for load-balancing between clouds) [DWC10].

Organizations use the hybrid cloud model in order to optimize their physical resources, to create high available infrastructures, and to increase their core competencies by margining out peripheral business functions onto the cloud, while controlling core activities on-premise through private cloud. Hybrid cloud has raised the issues of standardization and cloud interoperability.

Some advantages of Hybrid Clouds are the following:

- Security, since important data can be on private clouds, whereas general data can be stored in a public cloud. Therefore, companies control their physical infrastructure, and where their data is stored, based on its requirement and discretion. Moreover, companies internal people can define easily their security steps, such as firewalls, among others.
- Cost, since hybrid clouds result in cost-effective solutions for companies. Hence, companies maximize their private resources utilization, complementing them by using public clouds as needed. And this is cheaper than having the entire infrastructure in public clouds.
- High availability, since companies can have their private virtual infrastructure deployments replicated in public clouds, using active-active

(i.e, both infrastructures working together), or active-passive (i.e, the public infrastructure remains inactive until the private one fails) strategies to assure HA.

The main drawback of Hybrid Clouds are the extra effort that companies must assume to manage their infrastructure in different clouds. Hybrid clouds are likely to be configured by experts - instead of regular system administrators- with high networking skills to interconnect and manage VM appliances on multiple locations, and to deploy distributed applications.

2.3. Cloud federation

Cloud federation emerged with the lack of standards to communicate resources and services from different clouds within the hybrid cloud deployment model. In fact, hybrid cloud is a special case of Cloud federation.

The goal of Cloud federation is to enable cloud providers and IT companies to collaborate and share their resources [KTMF09] [RBE+11], to be able to fulfil requests during peak demands and negotiate the use of idle resources with other peers.

Cloud federation comprises resources and services from different providers aggregated in a single pool. This federation comprises many portability and interoperability features, such as resource migration (relocation of resources from one service domain to another domain), resource redundancy (similar service features in different domains), and combination of complementary resources services (different types to aggregated services). Service disaggregation is closely linked to Cloud Federation as federation eases and advocates the modularization of services in order to provide a more efficient and flexible overall system [KKB+11].

Cloud developers and researchers have proposed numerous federation architectures, including cloud bursting, brokering, and peering architectures [ea12]. These architectures can be classified according to the level of coupling or interoperation into: loosely coupled (with no or little interoperability among cloud instances), and tightly coupled (with full interoperability among cloud instances).

In this Section, we review the federation architectures and coupling levels, as well as the main challenges of Cloud federation. We focus on Cloud brokering principally, as this work is based on this type of federation.

2.3.1. Challenges of federation

The access to multiple cloud providers involves several challenges [Vou04] [DWC10] [GGB+15] that make the cloud usage difficult, such as different cloud interfaces, instance types, pricing schemes or image types.

The main challenges are explained below:

- Different cloud interfaces.

  Each cloud provider exhibits its proprietary interface, but some efforts have been done in order to standardize an interface for accessing to any cloud provider, such as the Open Cloud Computing Interface (OCCI) [NEPM11] [ME11], or APIs with various adaptors for accessing different clouds, such as DeltaCloud [18].

- Different instance types.

  Apart from standard instances, there are special High-CPU or High-Memory instances for high computing applications, also clustered instances if needed, or live-changeable instances depending on the provider. Nowadays, it is not easy to compare the performance of different instances in different clouds, which make difficult the optimization of cost or performance [LMZG15].

- Different pricing schemes.

  In the early phase of cloud adoption the price model was dominated by fixed prices [MT10] [FLL14]. Nowadays, cloud market trend shows that dynamic pricing schemes utilization is being increased, in which prices change according to demand in each cloud provider. Currently no pricing interface is available, so users find difficult to search cloud prices and decide where to put their resources [NID15].

- Different image types.

  Each cloud provider uses a particular image format. Thus, an image type created for example in Amazon EC2 (Amazon Machine Image, AMI) does not work in every provider, so users need to create an image type in almost each cloud provider.

  One possible solution is to contextualize the image, using in each cloud provider a pre-defined image and giving it a post- configuration script, which will prepare the image for working properly.

- Network latencies.

  In a multi-cloud environment, the challenge of how to cope with large amount of network communications among several virtual machines has to be addressed. Some user services can be critical in network communications having low tolerance to delays (e.g. live video streaming or parallel applications). Thus, the service will not get any benefit of a multi-cloud environment, unless these VMs that need to communicate between them are deployed in the same location.

[18] Delta Cloud home page - `http://deltacloud.org/`

- Network across clouds.

  Resources running on different cloud providers are located in different networks. However, some kind of services require all their components to be located on the same local network to communicate the different service components. This challenge can be addressed adopting the Virtual Private Network (VPN) technology to interconnect the different cloud resources in a secure way.

2.3.2. Federation architectures

In a federated scenario, cloud providers exhibit different degrees of coupling, regarding the cooperation between cloud instances, the level of control and monitoring over remote resources, the possibility of deploying cross-site networks, or even the possibility of migrating VMs between cloud instances [MVML12]. Before explaining federation architectures, we define the federation coupling levels:

- Loosely coupled federation.

  This scenario is formed by independent cloud instances - for example, a private cloud complementing its infrastructure with resources from an external commercial cloud - with limited interoperation between them. Usually, a cloud instance has little or no control over remote resources (for example, decisions about VM placement are not allowed), monitoring information is limited (for example, only CPU, memory, or disk consumption of each VM is reported), and there is no support for advanced features such as cross-site networks or VM migration.

- Tightly coupled federation.

  This scenario usually includes clouds belonging to the same organization and is normally governed by the same cloud management platform. In this scenario, a cloud instance can have advanced control over remote resources - for example, allowing decisions about the exact placement of a remote VM - and can access all the monitoring information available about remote resources. In addition, it can allow other advanced features, including the creation of cross-site networks, cross-site migration of VMs, the implementation of high-availability techniques among remote cloud instances, and the creation of virtual storage systems across site boundaries.

In practice, various federation architectures implement these coupling scenarios. Although there is no general agreement on the classification of these architectures, Figure 2.4 shows the three main types of federation architectures: bursting (hybrid), brokering, and peering. The combination of these architectures is possible to form other customized architectures.

While the loosely coupled hybrid and broker architectures have been widely studied and implemented, there is still much work to be done regarding the development and implementation of more coupled architectures, especially in the case of peering architectures [MVML12].

Bursting architecture.

This architecture refers to hybrid cloud deployment model that we introduced in Section 2.2.3. As Figure 2.4(a) shows, the cloud bursting architecture combines the existing on-premise infrastructure with remote resources from one or more public clouds, to provide extra capacity and satisfy peak demand periods.

Because the local cloud management platform has no advanced control over the virtual resources deployed in external clouds beyond the basic operations the providers allow, this architecture is loosely coupled. Most existing open cloud management platforms support the hybrid cloud architecture, which has been explored in various research efforts [Pet11] [SMLF09], and is used in infrastructures such as StratusLab (http://stratuslab.eu).

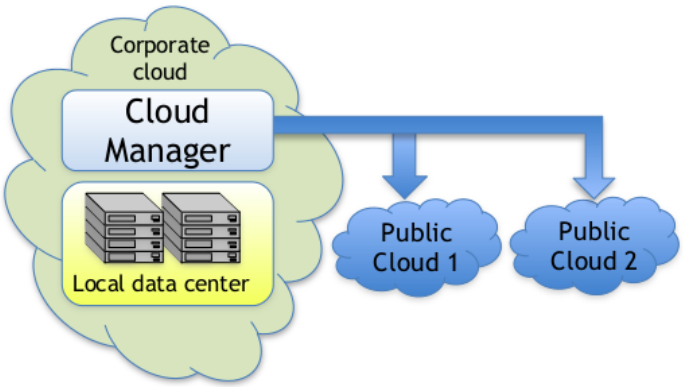
Broker architecture.

The central component of this architecture, shown in Figure 2.4(b), is a broker that serves various users and has access to several public cloud infrastructures. A simple broker should be able to deploy virtual resources in the cloud as selected by the user. An advanced broker, offering service management capabilities, could make scheduling decisions based on optimization criteria such as cost, performance, or energy consumption, in order to automatically deploy virtual services in the most suitable cloud, or it could even distribute the service components across multiple clouds. This architecture is also loosely coupled, since public clouds typically do not allow advanced control over the deployed virtual resources.

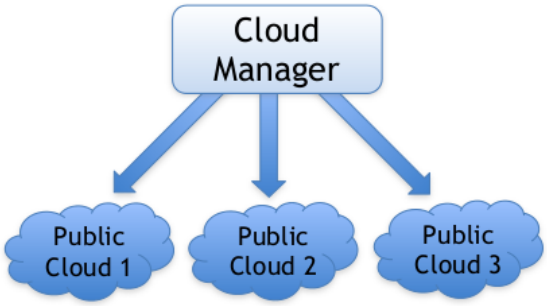
Brokering is the most common federation scenario, with many commercial and open source brokering services operating in the cloud market. In Section 2.3.3 we analyse in detail this architecture, since this work is based on it, and we also detail some examples of brokering architecture in research works, real infrastructures, and commercial tools.

Peering architecture.

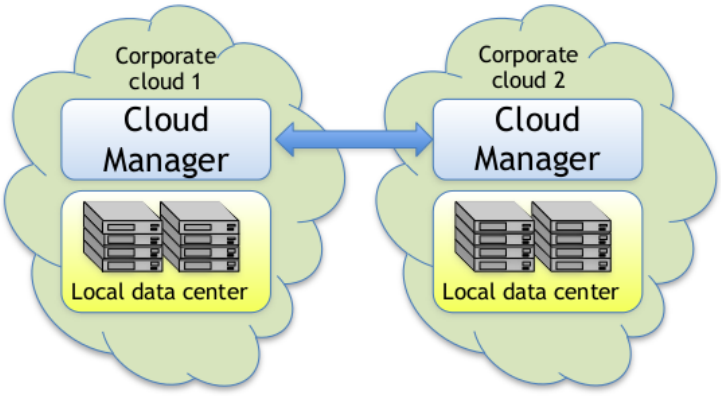
As Figure 2.4(c) shows, cloud peering architecture consists on two or more partner clouds that interoperate to aggregate their resources and provide users with a larger virtual infrastructure. This architecture is usually partially or tightly coupled, since partners could be provided with some kind of advanced control over remote resources, depending on the terms and condi-



(a) Cloud bursting (hybrid) architecture



(b) Cloud broker architecture.



(c) Cloud peering architecture

Figure 2.4: Cloud federation architectures

tions of contracts with other partners. These partner clouds usually have a higher coupling level when they belong to the same corporation than when they are owned by different companies that agree to cooperate and aggregate their resources. The Reservoir federated infrastructure is an example of an peering cloud architecture [RBE+11].

2.3.3. Cloud brokering.

In this section we explain the features of Cloud brokering, the federation model in which this work is based. As commented in Section 2.2, the current cloud market is composed of several public cloud providers, such as Amazon EC2 [19], Rackspace [20], or GoGrid [21]; private clouds, which are on-premise infrastructures managed by a cloud management platform, such as OpenNebula [22], OpenStack [23], Eucalyptus [24], or VMware vCenter [25]; and hybrid clouds [MGR11].

These cloud providers and platforms exhibit many differences regarding the functionality and usability of exposed cloud interfaces, the methods for packaging and managing images, the types of instances offered, the level of customization allowed for these instances, the price and charging time periods for different instance types, or the pricing models offered (e.g. on-demand, reserved, or spot).

To help cloud customers to cope with such a variety of interfaces, instance types, and pricing models, cloud brokers emerge as a powerful tool to serve as intermediary between end users and cloud providers [BYV+09] [ea12] [PR14]. A cloud broker provides an uniform interface independently of the particular cloud provider technology, and helps cloud users to choose the right options when multiple clouds are available [WNLL13] [WM14] [KAACF15] [LMZY15]. The main features that a cloud broker should provide to cloud users are the following:

- Intermediation: providing management capabilities atop an existing cloud platform.
- Aggregation: deploying customer services over multiple cloud platforms.
- Arbitrage: brokers supply flexibility, opportunistic choices, and foster competition between clouds.

[19] Amazon Elastic Compute Cloud (EC2) - `http://aws.amazon.com/ec2/`
[20] Rackspace - `http://www.rackspace.com/`
[21] GoGrid - `http://www.gogrid.com/`
[22] OpenNebula home page - `http://www.opennebula.org/`
[23] Open Stack Open Source Cloud Computing Software- `http://www.openstack.org/`
[24] Eucalyptus - `http://www.eucalyptus.com/`
[25] VMware Virtualization Software for Desktops, Servers and Virtual Machines for a Private Cloud- `http://www.vmware.com/`

However, most current cloud brokers do not provide advanced service management capabilities to make automatic decisions, based on optimization algorithms, about how to select the optimal cloud to deploy a service, how to distribute optimally the different components of a service among different clouds, or even when to move a given service component from one cloud to another to satisfy some optimization criteria. So, an open research line in Cloud brokering is the integration of different placement algorithms and policies in the broker for optimal deploying of virtual services among multiple clouds, based on different optimization criteria, for example cost optimization, performance optimization, or energy efficiency.

State of the art in cloud brokering.

In this section we review some of the current commercial, research, and open-source solutions. In the commercial side, some private companies offer brokering solutions in the current cloud market, such as RightScale [26], SpotCloud [27], Kavoo [28], or CloudSwicht [29], among others. In academia, there are several initiatives such as Mosaic or the Optimis project. And finally within the open-source world, in [Bro11] there is a list of cloud brokers and open source cloud management projects with a brief description of their offerings.

Here we review of some of the most used commercial solutions:

- *RightScale* offers a cloud management platform for control, administration, and life-cycle support of cloud deployments. It has an adaptable automation engine that automatically adapts the deployment to certain events in a pre-established way. In addition, it includes a multi-cloud engine that interacts with cloud infrastructure APIs and manages the unique requirements of each cloud.

- *Spot Cloud* provides a structured cloud capacity marketplace. In this marketplace the service providers sell the extra capacity they have and the buyers can take advantage of cheap rates selecting the best service provider at each moment. However, it does not perform this selection in an automatized way. It would be ideal if the user does not need to check the price of each cloud provider at each moment, and instead an optimization algorithm should be used to select the best way to place the VM according to the actual rates of all the cloud service providers.

- *Kavoo* provides application centric management of virtual resources in the cloud. It takes all the information that somehow affects the

[26] RightScale home page - `http://www.rightscale.com/`
[27] SpotCloud home page - `http://www.spotcloud.com/`
[28] Kavoo home page - `http://www.kavoo.com/`
[29] Cloud Switch home page- `http://www.cloudswitch.com/`

application and allows changing the operating system in which the application runs dynamically.

- *CloudSwitch* offers the possibility to run the applications in the best fitting instance among several cloud providers by comparing different instances with the requirements of the VM to be deployed. The main idea is to offer a solution to the heterogeneous and constantly expanding system that do not have an instance that fits all the necessities. To do so, it creates a comparative of different instances taking into account business and technical requirements and creates a fitting percentage that compares the VM necessities with the instances.

These commercial solutions still present some deficiencies for the cloud adoption, and hence many researches focus on the obstacles and opportunities that Cloud brokering presents nowadays. Some of these researches are at European level, and try to solve some of the problems commercial solutions have. Here is a short review of them:

- *Mosaic* [AFGJ10], that offers an open-source cloud application programming interface, that targets the development of multi-cloud oriented application, to offer a simple and transparent access to heterogeneous cloud resources, and avoid to lock-in into proprietary solutions.
- *Optimis project* [ea12], that offers a framework and a tool-kit to simplify service construction, support deployment and runtime decisions of hybrid clouds, and supports service brokerage via interoperability and architecture independence.
- In [BRC10] a federated inter-cloud environment is proposed, to avoid the actual obstacles and to be able to achieve all the Quality of Service (QoS) targets under variable workload, resource and network conditions. The redundancy and reliability needed to meet the QoS targets are obtained thanks to clouds in multiple geographical locations, and the performance targets are met dynamically resizing the resources. This work uses a simulated federation driven market and a list of not standardized protocols.

On the other hand, there are also open source brokering middle-ware available in the market, such as **Aeolus** [30], an open source, Ruby-written cloud management software sponsored by Red Hat, which runs on Linux systems. As a management software, Aeolus allows users to choose between private, public or hybrid clouds, using DeltaCloud cross-cloud abstraction library for making it possible. It has four different components: *Conductor*, which provides cloud resources to users, manage users access and use of

[30] Aeolus home page - `http://www.aeolusproject.org/index.html`

those resources, and control users instances in clouds. This lets users make intelligent choices about which cloud to use; *Composer*, which allows users to build cloud-specific images from generic templates, so that they can choose clouds freely using compatible images; *Orchestrator*, which provides a way to manage clumps of instances in an organized way. Users should be able to automatically bring up a set of different instances on a single cloud or spanning multiple clouds, configure them, and tell them about each other; and *HA Manager*, which provides a way to make instances or clumps of instances in the cloud highly available.

Regarding the provision of advanced cloud brokering services, Aeolus is not aware of pricing schemes or even of single prices, it does not include a scheduler to optimize deployments, so optimization algorithms cannot be used here, and it cannot run as a simulator, so every decision has to be made assuming real consequences.

2.4. Technology overview

In this section we explain, in a technical way and focusing on industry, how an institution can build its own private cloud, and, in case of using public clouds or hybrid ones, some tools that help institutions with the goal of deploying infrastructure in multiple clouds.

In the first case, we explain the steps to build a private cloud, focusing on Figure 2.1 layers. We review each layer mentioning its requirements, and also mentioning some companies which business is focused on these requirements. Moreover, we cite some open source tools which role is critical in this process.

In the second case, we explain the cross-cloud abstraction libraries used to deploy infrastructure on multiple clouds. Companies that have their applications in the cloud, generally use these libraries to be able to move their infrastructure deployments from one cloud provider to another, or to create everything from scratch in one or some providers and in an programmatic way. This way, these companies save money and time if something goes wrong with a certain cloud provider.

2.4.1. Building a private cloud

We consider important to explain the main factors required to build a private cloud infrastructure. Regarding the architecture shown in Figure 2.1, we focus on Hardware and Infrastructure layers to explain this requirements. To build a private cloud, it is not mandatory to include solutions for the other two layers of Figure 2.1.

First, we obviously need a ***physical infrastructure*** to set up the cloud. This infrastructure should be composed not only of several racks (or blades) of physical servers to handle computation, but also network element such

as switches and routers to handle network communication between physical servers, and between them and the Internet, and storage elements such as disk cabins to store images or manage storage backups. There are several well known physical infrastructure providers, such as HP [31], IBM [32], or Dell [33] for compute nodes, Cisco [34] or Juniper [35] for network products, or EMC [36] or NetApp [37] for storage solutions.

Once we have the aforementioned physical infrastructure, we need to install an ***hypervisor*** on every compute node. An hypervisor is a software installed on a physical machine that virtualizes physical hardware resources, and offers them to its virtual machines, and also it enables the creation, monitoring, and running of virtual machines. The computer in which the hypervisor is running is known as *host*, and each virtual machine is known as *guest*. There are several well known open-source hypervisors, such as KVM [38] - a virtualization infrastructure in the form of loadable kernel modules for the Linux kernel which turns it into a hypervisor-, or Xen [39]- free and open-source software, released under the GNU General Public License (GPL), born in The University of Cambridge Computer Laboratory, and maintained by The Xen community. Some companies have also developed its proprietary hypervisor, such as HyperV [40] - formerly known as Windows Server Virtualization, is a native hypervisor that enables platform virtualization on x86-64 systems-, or VMware vSphere [41] -the industry's first x86 "bare-metal" hypervisor-.

And finally, the last step is to install a ***cloud management platform***. Nowadays, there are some open projects that focuses on building a complete cloud management platform: OpenNebula [42], OpenStack [43] , or CloudStack [44] - Apache's proyect-.

- **OpenNebula** is an open-source project launched in 2005 as part of a research project, delivering a simple but feature-rich and flexible solution to build and manage enterprise clouds and virtualized data

[31] HP Official Site - `http://www.hp.com/country/us/en/hho/welcome.html`
[32] IBM Official Site - `http://www.ibm.com/us/en/`
[33] Dell Official Site - `http://www.dell.com/`
[34] Cisco Systems - `http://www.cisco.com/`
[35] Juniper Networks - `http://www.juniper.net/us/en/`
[36] EMC Data Management Solutions and Storage Hardware Products - `http://www.emc.com/index.htm?fromGlobalSiteSelect`
[37] NetApp Data Storage - `http://www.netapp.com/us`
[38] Kernel Based Virtual Machines - `http://www.linux-kvm.org/page/Main_Page`
[39] Xen Hypervisor home page - `http://xen.org/`
[40] Microsoft HyperV Server home page - `http://www.microsoft.com/oem/es/products/servers/Pages/hyper_v_server.aspx`
[41] VMware Virtualization Software for Desktops, Servers and Virtual Machines for a Private Cloud- `http://www.vmware.com/`
[42] OpenNebula home page - `http://www.opennebula.org/`
[43] Open Stack Open Source Cloud Computing Software- `http://www.openstack.org/`
[44] Apache CloudStack - Open Source Cloud Computing - `http://cloudstack.apache.org/`

centers. It is subject to the requirements of the Apache License version 2, combines existing virtualization technologies with advanced features for multi-tenancy, automatic provision and elasticity, following a bottom-up approach driven by the real needs of sysadmins and devops. It orchestrates storage, network, virtualization, monitoring, and security technologies to deploy multi-tier services as virtual machines on distributed infrastructures, combining both data center resources and remote cloud resources, according to allocation policies.

- **OpenStack** is an open-source project launched in July 2010 by Rackspace [45] and NASA intended to help organizations offer cloud-computing services running on standard hardware. It has a modular architecture with various code names for its components, such as *Nova* - a Cloud computing fabric controller -, *Swift* - a scalable redundant storage system-, *Cinder* - persistent block-level storage devices for use with OpenStack compute instances-, *Neutron* - system for managing networks and IP addresses-, *Horizon* - graphical interface to access, provision and automate cloud-based resources-, *Keystone* -central directory of users mapped to the OpenStack services they can access-, *Glance* - discovery, registration and delivery services for disk and server images-, *Heat* - a service to orchestrate multiple composite cloud applications using templates-, and *Ceilometer* - a Single Point Of Contact for billing systems, providing all the counters they need to establish customer billing, across all current and future OpenStack components.

2.4.2. Cross-cloud abstraction libraries.

When talking about hybrid clouds, avoiding vendor lock-in, distributed applications with different levels of coupling, federated architectures, or centralized management of heterogeneous providers offerings, it seems to be hard to deal with everything in a programmatic way.

For that purpose, there are code developments called cross-cloud abstraction libraries, that help developers to abstract from every different cloud feature, manage distributed infrastructure in a readable way, and give them more flexibility to provide value to their applications.

Distributed cloud infrastructures management is not closed to any programming language, but some of them have their own cross-cloud library, such as jClouds library for Java developers, or Fog library for Ruby developers, among others. Moreover, there are also efforts from academia and private companies towards this direction, such as DeltaCloud from RedHat.

Some of the first Cloud brokering solutions of the market are just web front-ends using these kind of libraries for the back-end. Here we explain some of these libraries:

[45] Rackspace - `http://www.rackspace.com/`

DeltaCloud.

DeltaCloud, a top-level Apache project, is an API developed by Red Hat that abstracts the differences between clouds. Some clouds supported by DeltaCloud are Amazon EC2, GoGrid, RHEV-M, etc. or OpenNebula-built clouds [SSMMLF08] [MVML11a].

Deltacloud enables management of resources in different clouds by the use of one of three supported APIs. The supported APIs are the Deltacloud classic API, the DMTF CIMI API, or even the EC2 API. Deltacloud maintains long-term stability for scripts, tools and applications and backward compatibility across different versions.

JClouds.

Apache JClouds is an open source multi-cloud tool-kit for the Java platform, that gives developers the freedom to create applications that are portable across clouds while giving them full control to use cloud-specific features [46]. It supports cloud providers such ash Amazon, Azure, GoGrid, OpenStack, Rackspace, or Google.

JClouds provides different interfaces such as the followings: *Compute*, that allows users to provision their infrastructure in any cloud provider and control the entire process, i.e, deployment configuration, provisioning and bootstrap; *BlobStore*, that allows users can easily store objects in a wide range of blob store providers, regardless of how big the objects to manage are, or how many files are there; *Load Balancer*, that provides a common interface to configure the load balancers in any cloud that supports them, just defining the load balancer and the nodes that should join it; *Specific APIs*, such as DNS, firewall, storage, configuration management, or image management, among others.

Fog.

Fog is the Ruby cloud services library that provides an accessible entry point and facilitates cross service compatibility. It is licensed under the MIT License, and available from its GitHub [47] public project which has been developed and is constantly improved by the community.

Fog works with *collections*, which provide a high level simplified interface to each cloud, making clouds easier to work with and switch between; *requests*, which allow power users to get the most out of the features of each individual cloud; and *mocks*, which make testing and integrating a breeze.

Finally, as the reader might imagine, testing code using real clouds can be slow and expensive, constantly turning on and shutting down instances.

[46] Apache JClouds home page - `https://jclouds.apache.org/`

[47] Fog - The Ruby cloud services library - `http://fog.io/`

Fog provides mocking, which allows skipping this overhead by providing an in memory representation resources as you make requests.

Chapter 3

Virtual infrastructure management techniques in multi-cloud environments

If you were plowing a field, which would you rather use, two strong oxen or 1024 chickens?
(Si tuvieras que arar un campo, ¿qué preferirías usar, dos fuertes bueyes o 1024 pollos?)

Seymour Roger Cray

In Chapter 2 we explained the Cloud federation and Cloud brokering concepts, together with different features of current public cloud providers. We realized that there is a gap between current Cloud brokering state and an ideal state, in where resources could be delivered to cloud users in an automated way, optimizing their deployments, or minimizing their investment. Therefore...

...the main purpose of this thesis is to research different mechanisms to support Cloud brokering, and to provide advanced placement algorithms for VMs that optimize a user criteria based on infrastructure parameters.

In this Chapter we describe the proposal of this thesis, and it is organized as follows:

1. We explain in depth the Cloud brokering architecture used in this work, its components (e.g, database, cloud manager, information manager, and scheduler), and how broker users interact with this architecture.

We focus specially on the scheduler, and also on how a service is defined within the architecture, showing how users can describe the number of virtual machines they need, or the optimization criterion they want to use, among other features.

2. We present the main brokering algorithms used this work. First, we introduce the strategies and scenarios that we consider in this work. Among the strategies, we propose cost optimization and performance optimization strategies. Among the possible scenarios, in order to simulate the real world, we work with static and dynamic scenarios.

   Next, we formulate the mathematical notation used in our algorithms. We mainly divide the algorithms part in the following: objective formulation, in which we define the parameter to optimize and the equations to follow; and constraint formulation, in which we define the constraints to these optimization equations.

3. Regarding the dynamic pricing schemes, we explain why we need to forecast prices, and the inclusion of forecasting algorithms within the scheduling module. For that purpose, we propose an estimation algorithm, and later we dig into forecasting algorithms literature, in order to compare and improve estimations in this work.

4. Finally, we extend our previous proposal of brokering algorithms to consider storage costs as a critical parameter in our objective of deploying infrastructure. We propose some storage strategies that we divide into uploading, deleting, and transferring images from the user placement to the cloud provider.

3.1. Brokering architecture

In this work, we propose a cloud broker architecture for deploying and managing infrastructure resources and services as introduced previously. It can be depicted in Figure 3.1. We mainly focus on the implementation of a dynamic scheduler for multi-cloud brokering environments, based on this architecture.

The main technical features of this cloud broker architecture are the following: modular, since it provides basic components but allow them to be easily replaceable by others (i.e. scheduling policies); open, since its code is planned to be available for developers and the scientific community with an open source licence; adaptable, since several use cases can be adapted using the cloud broker; and based on standard interfaces.

Three main actors interact with the proposed cloud broker architecture: the broker administrator, the user of the broker, and the cloud providers.

The administrator is responsible for configuring the cloud broker. That includes the definition of a cloud provider list, with their corresponding accounting information, and an instance list that includes the available instance types in each cloud and their prices.

The user receives information of both cloud and instance lists, and requests the deployment of a given service. A service is a set of components, each one composed by a number of virtual machines, a common optimization criteria, and some particular restrictions. All of these service description options are included in a service description file, and are detailed below.

At the same time, cloud providers can offer different kind of resources associated to particular pricing schemes.

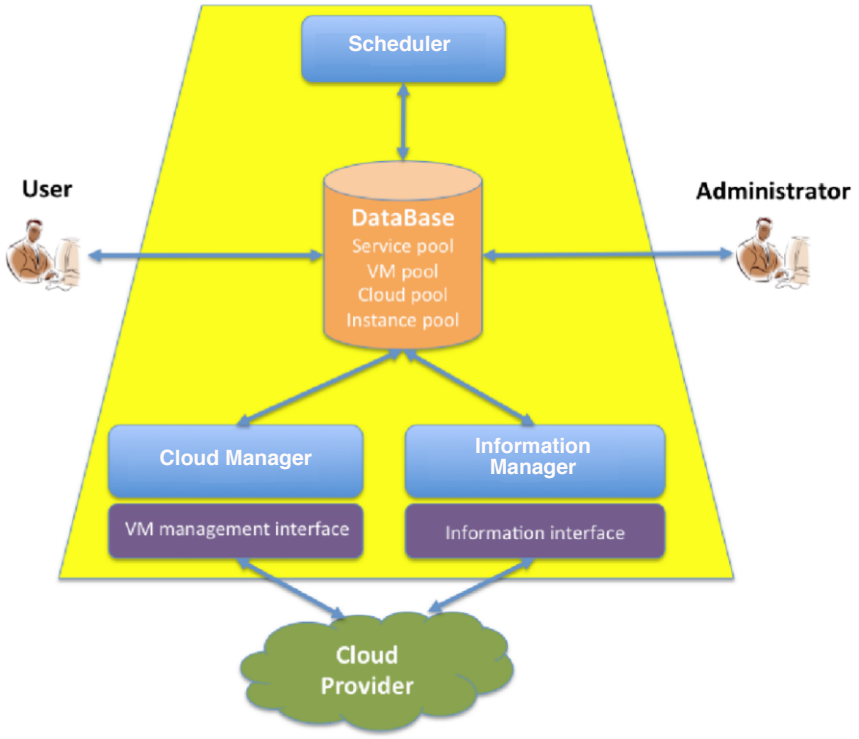


Figure 3.1: Cloud Brokering Architecture Overview.

The **cloud broker** is an intermediary between users and providers. It performs two main actions:

- Placement of virtual resources of an infrastructure across a set of cloud providers according to user optimization criteria.
- Management and monitoring these virtual resources.

In order to perform these two actions, the cloud broker is divided into three

different components: the Scheduler, the Cloud manager, and the Information manager.

The **Scheduler** is responsible for taking the placement decision, which can be addressed using a static or dynamic approach. The static approach occurs when deployment conditions do not change. The dynamic approach is suitable for variable conditions (e.g, variable resource prices, required virtual resources, or cloud provider resources availability). We have implemented this scheduling module for managing dynamic pricing situations.

In order to achieve an optimal deployment of virtual resources according to user criteria, the scheduler can use several algorithms which are described in the next section. These algorithms should run periodically to adapt the resource deployment to the variable pricing conditions.

The scheduler output is a deployment plan, which is composed by a list of VM templates. Each template includes the target cloud provider to deploy the VM in, and some attributes for the selected provider. The deployment plan is used as an input for the cloud manager component.

The **Cloud manager** addresses the management and monitoring actions using the *OpenNebula (ONE)* cloud management platform [SMLF09]. ONE provides an uniform and generic user interface to deploy, monitor and control VMs in a pool of distributed physical resources.

The use of specialized adapters is a well-known technique to achieve a multi-provider interoperation. In general, OpenNebula is able to interoperate with some cloud providers, like Amazon EC2 [1], ElasticHosts (EH) [2] or other clouds compatible with Deltacloud-based API [3]. These adapters convert the general requests made by the virtualization components of ONE to manage VMs through the respective APIs.

Finally, the **Information manager** is responsible for gathering cloud providers static information, and pricing and instance availability information dynamically.

Technical details

Focusing on Figure 3.1, here we explain some technical details of each component of the architecture, and the way we plan to use them:

- Database.
  The database back-end is the central component of this architecture. It stores the cloud, instance, service, and virtual machine lists, which are used by the rest of architecture components.

  The cloud list stores information about cloud providers. Each provider

[1] Amazon Elastic Compute Cloud (EC2) - `http://aws.amazon.com/ec2/`
[2] ElasticHosts - `http://www.elastichosts.com/`
[3] Delta Cloud home page - `http://deltacloud.org/`

is linked to a file with information about accounting information (such as user name, passwords, or public and private key paths).

The instance list stores information about the available type of instances in each cloud. Each instance belongs to a cloud and is defined by an instance type (e.g. small or large), with its price and the pricing model (e.g. on- demand, reserved, or spot models) to be applied.

The service list stores information about the services defined by the user. Each service is described in a service description file.

And finally the VM list stores information about VMs managed by the cloud broker in different clouds. Each VM belongs to a service and is mapped to a particular instance type. It also includes a current status (pending, running, cancelled, shut-down or failed) and some timing and resource consumption information (e.g. CPU, memory, or network usage).

- Scheduler.
  The scheduler reads the service list, and uses the service description file as input to deploy each new service.

  It invokes to the particular scheduling module specified in the service description (once for static scheduling, and periodically for dynamic scheduling), then it decides a set of VM to be deployed (or cancelled) in each cloud, and updates the VM list to inform the Cloud manager which VM must be deployed/cancelled.

  Before each scheduling action, the scheduler also reads the instance list to know the type of instances available in each cloud, the price of each instance, the number of instances available, and certain deployment information.

- Cloud Manager.
  The Cloud manager periodically reads the VM list, submits the VM in pending state, and shut-downs the VM in cancelled state. When a new submitted VM is deployed and running in the cloud, or a cancelled VM is shut-down, the Cloud manager updates its state in the VM list. The Cloud manager also monitors periodically the deployed VM, and collects data about CPU, memory and network usage of each one. This information is updated in the VM list. The Cloud manager uses the accounting information in the cloud list to access to each cloud in order to deploy/shut-down/monitor VMs.

- Information Manager.
  The information manager periodically collects information about instance availability and instance prices for all the instances listed in the instance list. Then it updates the instance list with the information

collected. It is specially useful in dynamic prices case, in which it is necessary to periodically update these prices.

- Service description.
  The service description is a file defined by the user in some standard language (e.g. XML or Json), which contains detailed information about the service to deploy by the broker, such as the components of the service, optimization criteria, scheduling policies to use, scheduling constrains, type of instances to use, performance of each instance, and so on.

  The information sections of the service description are the following:

  - Service components and lifetime.
    This part of the service description includes a list of components of the service, which is an enumeration of the components that will be deployed as VM in different clouds (e.g. Component 1: web server front-ends; Component 2: data-baser servers; Component 3: application servers (back-ends); Component 4: file server); a list of images (e.g. Amazon Machine Image -AMI- in Amazon EC2) associated to each component in each cloud to use; a list of post-configuration files for each service component (if necessary); and timing information (e.g. service start and end times).

  - Scheduling parameters.
    For each service component, we must specify the scheduling parameters we want to use for scheduling and deploying this component. These parameters are: scheduling strategy, which can be static or dynamic; scheduling period, which is the interval between consecutive scheduling decisions in dynamic scenarios; scheduling policy, driven by various optimization criteria and different kinds of restrictions, both detailed in Section 3.2.4.

  - Cloud/Instance usage and instance performance.
    The user can define which clouds (among those available) wants to use for deploying a given service component, and which kind of instances wants to use. In addition, the user can also specify the performance that each instance type is offering for his particular service (notice that the performance analysis of each instance must be done off-line by the user, and provided as an input of the service description).

Once explained how we design the broker, in next section we explain in depth the strategies and algorithms we design and implement in this work.

3.2. Brokering algorithms

As commented in Section 3.1, the broker is designed to accept different brokering algorithms for deploying virtual infrastructure among multiple clouds. Each algorithm follows a particular deployment strategy. In this section we explain the brokering strategies designed in this work, and the possible scenarios to perform optimal deployments.

3.2.1. Strategies and scenarios.

The two main strategies we consider in this work are based on two different **optimization criteria**, concretely to optimize infrastructure cost or to optimize application performance.

Cost optimization strategy is about deploying the infrastructure in the cheapest placement, once having retrieved the pricing information of every available public cloud. This placement can be in a single cloud - if there are available resources of different instance types, and if the cheapest resources are in the same cloud -, or different clouds - if cheapest types of instances are divided among different clouds, per type of instance. In this strategy we do not take into account other parameters, such as the physical distance between different tiers within an application (and therefore the possible lag between communications).

Performance optimization strategy is about deploying the whole infrastructure using those instance types in which the particular application to be deployed performs better, once having tested this application on different types of instance. Here, for instance, prices are not taking into account by default, so the total cost of the infrastructure could be excessive. We address the cost issue by adding some constraints, which in turn add some complexity to the strategy.

We formulate these strategies in mathematical notation in Section 3.2.2, together with some restrictions that help to concrete which is the goal we would like to achieve.

About **brokering scenarios**, we consider the following scenarios to deploy our infrastructure: static and dynamic scenarios.

Static scenarios are those in which neither cloud user nor cloud provider conditions change along time. This is to say that there will be an unique deployment election considering a static snapshot of the public cloud providers offerings. Nothing else is considered in this scenario except the deployment strategy selected. For instance, conditionals such as application lifetime, variable workloads, or variable prices do not affect in this scenario.

Dynamic scenarios are those where conditions can variate along time. The best example of changing conditions are variable prices. However, other conditions may also change, such as resource availability, application perfor-

mance, or others less frequent such as new regions. Here we explain all of them:

- Prices.
  The variability of prices is usually caused by the unpredictable demand of the cloud resources, in the cloud providers side (i.e, hardware infrastructure occupation demand), not in the cloud user side (i.e, application demand). For example, Amazon uses dynamic spot prices to sell its idle computing resources at lower prices, so trying to gain more users. In this case, the more occupied their infrastructure is, the less spot instances will be available.

  Each particular cloud provider updates its instance types prices in different periods of time (e.g. 5 minutes, 30 minutes, 1 hour, or other particular periods). As we consider Amazon EC2's spot prices as the dynamic pricing scheme to use in our experiments, we are aware of that Amazon updates prices each 30 minutes. However, we consider a hourly period for obtaining new prices because it is the minimum charging period that Amazon applies.

- Resource availability.
  Resource availability depends on public clouds resources demand, and also on available physical infrastructure. Most cloud providers map different type of hardware that they have internally (for example, certain machines with processors at certain speed) to different type of resources, named *families* (e.g, *m1* family, which is formed by m1.small, m1.large, and m1.xlarge types of instance, all of them offered by AWS). In other words, one family of types of instances usually runs in the same type of physical machines.

  Nowadays, even Amazon, that has a huge hardware infrastructure that appears to be infinite to cloud users, is expected to update their hardware once in a while, when new hardware technology appears in the market (e.g, powerful processors or faster hard disk units). When cloud providers update their hardware infrastructure, they usually start purchasing this new hardware and stop purchasing the oldest (or the worst regarding cost/benefit) type of hardware, being this moment the begin of the deprecation of this hardware.

  This update action results in offering new types of instances (e.g, m2.large, where *m2* is the name of the new family of resources) in which, for example, Amazon offers solid state disks (SSD) instead of traditional magnetic disk. Obviously, old resources like m1 family can not be hosted in the new hardware infrastructure, and their availability is expected to decrement or disappear along time.

- Application performance.

Not only cloud providers conditions may change, but also user's application requirements may change. For instance, in a classical three tier application, client request load is unknown in advance, so it can change several times during any period of time. This is the typical case of unpredictable user demand. When the demand changes and the virtual machine starts to get overloaded, it could affect to some of these tiers in different moments of the service lifetime.

- Others.
  Following the Amazon EC2 case, they build new regions as their resource demand grows. It happened some time ago with the Asia region [Anu10], and in 2014 they have set up a new data center in Frankfurt [Anu14]. Focusing on prices, when a region appears, its infrastructure resource prices are not the same as other regions as everyone can observe in Amazon's web page, so we need to include the new region and features in the broker. As new regions or disruptive features do not appear very frequently, the broker is designed to be changed by the administrator but in a manually way.

In this work, we consider price variations as the main effect of dynamic scenarios, although we also experiment with application performance variations. We explain different pricing schemes of public cloud market in Section 2.2.2, but to highlight them here we summarize the three available types of pricing schemes:

- On demand, in which cloud users pay as they use resources, and nothing else. In general, cloud providers have their own non-standardized billing method (i.e, per hour, the first hour and then per periods of 10 minutes, and so on). Within this scheme, we divide these resources into:
  - Pre-defined: The classic resources offered by cloud providers. They are made of certain quantity of cores, RAM and disk, and it is not possible to modify them.
  - Customized: When providers allow users to configure their own resources by modifying the amount of cores, memory or disk. Not every provider has this type of resource.
- Reserved, in which cloud users reserve resources for a long period of time. They pay in advance an amount of money (i.e, per six months, or per year) and then the price per hour is cheaper than on-demand scheme. Most providers offer this scheme.
- Spot, in which the price fluctuate along time. This is the cheapest one, and it is billed in the same way as on-demand, but not every provider offers this scheme.

We must take these pricing schemes into account, since the infrastructure to deploy could be active for hours, days, or even months, so the brokering decision would change in case that, for example, we do not consider reserved instances.

3.2.2. Mathematical formulation.

Once we defined the strategies and scenarios considered in this work, we detail the proposed scheduling algorithms for optimal deployments in both static or dynamic cloud pricing scenarios.

In each scenario, the goal is to deploy a service composed of a set of components which are executed as virtual machines in different cloud providers. Thus, in this work we deploy a number n of VMs, $v_1, ..., v_n$, that belong to certain instance types it, $it_1, ..., it_l$ it_j , across the m available clouds, $c_1, ..., c_m$ to optimize user criteria, such as cost or performance. It is important to notice that in this work the number of VMs (n) can be fixed or dynamic to cover different situations, e.g. if users demand change, or if an big instance type gives worst performance than certain number of smaller ones, and it should be replaced by them. Also in dynamic scenarios we take into account the variability of prices in each cloud provider.

In this work we consider periods of one hour for dynamic scenarios. Therefore, we define *scheduling period* as the next one-hour period to schedule. Hence, we execute the broker before the beginning of each scheduling period, which causes a total or partial reconfiguration of the virtual infrastructure.

Each cloud provider offers a given number instance types, which are composed of an amount of cores, RAM memory, and disk storage, regarding a minimum and maximum quantity of each component. Other providers allow the user to configure his or her own instance type modifying the predefined amount of cores, memory or disk. However, in this work we mostly use a set of instance types offered in most of the available clouds, in order to work with the same pool of instance types and facilitate the use cases understanding.

Moreover, in this work we consider using different combinations of instance types: in some experiments we use a single type of instance in order to deploy an homogeneous infrastructure, but in other experiments we use a mixture of instance types to deploy a heterogeneous infrastructure.

To understand better the experimentation part of the rest of the work, mainly Chapter 4 experiments, we introduce some useful definitions:

- We define t as any one-hour period.
- We consider a $0-1$ integer programming formulation where $X_{i,j,k}(t) = 1$, $1 \leq j \leq l$, $1 \leq k \leq m$ if virtual machine v_i that belongs to instance type it_j is placed at cloud c_k during period t, and 0 otherwise.

- Using t, we define $P_{j,k}(t)$, $1 \leq j \leq l$, $1 \leq k \leq m$ as the real price paid for a virtual machine that belongs to instance type it_j deployed in cloud c_k during period t.

Once we set up the basics of our algorithms, we move on towards the formulation of the objective of each scheduling strategy.

3.2.3. Objective formulation

To explain deeper the aforementioned brokering strategies, in this work we choose integer lineal programming. In a linear program, the objective function and the constraints are linear relationships. Many real-world business problems can be formulated in this manner, providing a powerful and robust analytical methodology for supporting fact-based decision making.

To solve experiments of Sections 4.1 and 4.2, we chose AMPL language [FGK90] due to its ease of use and its similarity with mathematical notation. AMPL can be used with a range of back-end solvers [FGK90]. Our first choice is the well-known MINOS and CPLEX [4] solvers.

However, in the final stage of this work (experiments of Section 4.3), we extend and use a Java simulator in which we adapt the needed AMPL library for Java, called *Choco* [5], to perform our simulations.

Cost optimization policy

In this approach we want to minimize the cost of each virtual machine that belongs to a certain instance type, by choosing the cloud which exhibits the lowest prices for this instance type. In general, the cost function we want to minimize is the Total Infrastructure Cost (TIC(t)) which is defined as the sum of the cost of each virtual machine in a given period of time:

$$TIC^t = \sum_{i,j,k}^{n,l,m} X^t_{i,j,k} \cdot P^t_{j,k} \tag{3.1}$$

being $X^t_{i,j,k}$ and $P^t_{j,k}$ as defined before. By minimizing equation (3.1) we address the challenge of deploying virtual resources in the cheapest placement.

As commented before, we identify two separate scenarios, static and dynamic. We always use equation 3.1 as our principal model, but depending on the scenario we vary the definition of $P^t_{j,k}$ to adapt this equation to these scenarios. This variation depends on the pricing schemes considered, which we explain in Section 3.2.1.

[4] IBM Corporation. ILOG CPLEX - `http://www.ilog.com/products/cplex/`

[5] Choco Solver - `http://choco-solver.org/`

In *static scenarios*, we address the issue of optimizing infrastructure costs by deploying the VMs once, taking advantage of the best cloud offers. As prices in static scenarios are fixed and known before each deployment, we consider all pricing schemes except the spot pricing one, because of its dynamic nature. We consider the following definition of $P^t_{j,k}$ in static scenarios:

$$P^t_{j,k} = \begin{cases} c_j & if \ \ pre-defined\ instance\ type \\ (CPU's \cdot c_{cpu}) + \\ (RAM \cdot c_{ram}) + & if \ \ customized\ instance\ type \\ (HDD \cdot c_{hdd}) \\ c_{j,rsv} & if \ \ reserved\ instance\ type \end{cases} \tag{3.2}$$

In *dynamic scenarios*, prices of similar instances change along time because of dynamic demand. As these prices are unknown, or as they fluctuate along each scheduling period, we use some prediction techniques based on historical information and propose them in Section 3.3.

Therefore, to formulate these alternatives, we define the following:

- Estimated equation.

  We define the estimated equation because user does not know prices for next hour because unpredictable dynamic demand makes prices change. In fact, prices of cloud resources in period t are unknown until period t finishes. Therefore, we need estimation mechanisms in order to select correct clouds based on estimated prices, and using the scheduler. Hence, we define the estimated total infrastructure cost (TIC_{est}) as:

$$TIC^t_{est} = \sum_{i,j,k}^{n,l,m} X^t_{i,j,k} \cdot E^t_{j,k} \tag{3.3}$$

  where $E^t_{j,k}$ is the estimated price if we deploy a virtual machine v_i of instance type it_j in a cloud c_k in period t.

- Oracle equation.

  To check the quality of the estimated prices, we introduce a new figure that we call *oracle price*, which is the best real price of each instance type for the next scheduling period, which is not known until the end of such period, considering all available clouds c_k. If we could know these prices, we would act as an oracle for taking decisions, getting the optimal ones. We define the oracle price of instance j (O_j) as follows:

$$O^t_j = \min_{1...m} P^t_{j,k} \tag{3.4}$$

where $1 \leq k \leq m$.

Next, we need to get the best oracle deployment, which is composed of the best combination of it_j oracle prices (O_j). Therefore, we need to minimize the Oracle equation, which we define as it is shown in 3.5.

$$TIC^t_{Oracle} = \sum_{i,j,k}^{n,l,m} X^t_{i,j,k} \cdot O^t_{j,k} \tag{3.5}$$

where:

$$O^t_{j,k} = \begin{cases} O^t_j & if\ the\ cheapest\ instance\ type\ is\ in\ cloud\ k \\ \infty & otherwise \end{cases} \tag{3.6}$$

In short, we define $O^t_{j,k}$ as the best price of a virtual machine that belongs to instance type it_j deployed in any cloud in period t. If there are more than 1 provider, we set two possibles values to $O_{j,k}$: *the cheapest price*, or infinity (to avoid choosing this cloud).

Performance optimization policy

In this approach we try to maximize the performance of the entire infrastructure by placing each virtual machine in the cloud which gives the highest performance. The performance function to maximize is the Total Infrastructure Performance (TIP(t)) which is defined as the sum of the performance of each virtual machine in a given period of time. As with the cost optimization policy, we also separate static from dynamic scenarios.

Hence, we add the following definition to the previous ones:

- We define $Perf_{j,k}$, as the performance of a virtual machine belonged to an instance type j deployed in cloud k.

The performance $Perf_{j,k}$ of an application depends on several factors:

- the type of instance used, since each instance type has its particular features (see table 2.1), or the physical infrastructure where the instance type runs, since different hardware features can also make the performance different.

- the requirements of the application (such as memory usage, cache use profile, disk I/O, or CPU utilization), since each application performs different when it runs on top of a single instance type.

Therefore, the owner of the application should provide this performance information once he/she tests the application. Ideally, this testing exercise should be done for each type of instance within every public cloud provider that he/she considers, and for each application or workload.

Some examples of infrastructure performance are *FLOPS* (floating operations per second) in case of high performance clusters, or *number of cores* in case of parallel computing. Some examples of application performance information are *request per second* in case of web servers, *reads or writes per second* in case of databases, or *messages sent or received per second* in case of message queues. Both application and infrastructure metrics can be used in $Perf_{j,k}$.

The equation to maximize in this case is:

$$TIP^t = \sum_{i,j,k}^{n,l,m} X^t_{i,j,k} \cdot Perf^t_{j,k} \tag{3.7}$$

3.2.4. Constraint formulation

The previous scheduling policies can be associated with different types of constraints. Constraints control how the broker deploys virtual resources, to reach certain objectives but without optimizing them. For example, cost optimization policy with a minimum performance constraint to reach, or performance optimization policy with certain instance type constraint to control which types we want to use. Therefore, we define constraints related to reach certain objectives, such as cost or performance, as follows:

Cost

The performance optimization strategy can work with a cost constraint, which can be expressed as follows:

- *Cost constraint*:

$$TIC^t \leq Cost^t_{max} \tag{3.8}$$

This means that the performance of a virtual infrastructure must be optimal but without exceeding a given cost threshold in each moment (t). To understand better the cost constraint, here we mention a typical case:

- Maximize our web server performance, but not exceeding our budget of 10 € per hour. Therefore, $Cost^t_{max} = 10$.

Performance

The cost optimization strategy can also work with a performance constraint. This means that the TIC must be optimized, but reaching a minimum

performance in each moment (t). In other words, the infrastructure has to provide users with a certain quality of service.

Performance can be measured using several *kpi's* (key performance indicator), such as number of CPU cores, MFLOPS, Gb of RAM, or application particular indicators.

To understand better the performance constraint, here we explain two examples of *kpi's*:

- Optimize a cluster infrastructure cost, but guaranteeing at least 10 MFLOPS of performance.

- Optimize a web server infrastructure cost, but guaranteeing enough performance for at least processing 100 requests per hour.

If we do not use performance constraint the broker probably finds a cheapest deployment, but using it we assure certain quality of service in our deployments. Moreover, we define another constraints related to the VM scheduling process, such as reallocation, instance type, or placement constraints.

Reallocation

We need a reallocation constraint to cope with the problem of temporary system performance degradation, which is one of the challenges of dynamic resource allocation.

In dynamic scenarios, in each scheduling period, part or event the whole infrastructure can be moved from one cloud to another cloud location. This reallocation action causes some virtual machines to be stopped during a short period of time, and it results in a temporary infrastructure performance degradation.

In this work, we use the reallocation constraint as a solution for assuring a certain system performance. The scheduler can prevent users from moving the whole set of resources but only moving some part of them, keeping the rest unchanged. So, we define this constraint as follows:

- *Reallocation constraint*:
  It provides the possibility of reallocating only a certain number of *kpi's* in each scheduling decision. It is useful when it is critical to keep part of the virtual cluster working without stop in order to guarantee a certain number of *kpi's* working in a certain moment (e.g. cores). Moreover, it allows to control service performance degradation while saving some money by taking advantage of dynamic pricing.

$$R^t_{min} \leq R^t \leq R^t_{max} \tag{3.9}$$

In Equation 3.9, R_{min} and R_{max} refer to the minimum and maximum number of *kpi's* that the scheduler can reallocate. *Reallocation* is defined as the difference between the last deployment performed and the next deployment to perform, in terms of number of *kpi's* deployed in each cloud. For that purpose, the cloud broker compares the current placement of the virtual machines with the new one.

$$R^t = \frac{\sum_{i,j,k=1}^{n,l,m} \left| X_{i,j,k}^t - X_{i,j,k}^{t-1} \right| \cdot kpi_j}{2} \tag{3.10}$$

In Equation 3.10, reallocation parameter is divided by two because it is only taken into account the number of *kpi's* to start in a new cloud. In other words, Equation 3.10 means the number or *kpi*ś to move across clouds.

To understand better the reallocation constraint, here we mention some typical cases:

- Allow total reallocation of the infrastructure. Therefore, $R_{min} = 0$, $R_{max} = 1$.
- Reallocate up to 20 % of the infrastructure for bursting purposes, keeping the rest in the same placement for giving users a proper quality of service. Therefore, $R_{min} = 0$, $R_{max} = 0,2$.

Instance Type

To deploy an homogeneous or heterogeneous infrastructure, we need a restriction to control the allowed types of instance to use:

- *Instance type constraint*:
  It provides the possibility to use only a certain type of virtual machines in each deployment.

  In Equation 3.11, it_{min} and it_{max} refer to the minimum and maximum percentage of the instance types to use in each deployment.

$$it_{min}(j) \leq \frac{\sum_{i,k}^{n,m} X_{i,j,k}}{n} \leq it_{max}(j), \quad 1 \leq j \leq l \tag{3.11}$$

To understand better the instance type constraint, here we mention some typical cases:

- Select *large* instance types for at least 50 % of the infrastructure. Therefore, $it_{min}(j) = 0,5$.

- Avoid using *extra large* instance types because its ratio cost/performance is not optimal. Therefore, $it_{min}(j) = 0$, $it_{max}(j) = 0$.
- Only use *small* instance types for the infrastructure since the application do not take advantage of multi-core instance types. Therefore, $it_{min}(j) = 1$, $it_{max}(j) = 1$.

Placement

To deploy an infrastructure in a static or a dynamic scenario, we need a restriction to control the allowed cloud providers to place the infrastructure in:

- *Placement constraint*:
  It provides the possibility to maintain a certain number of VMs in each cloud placement.

  In Equation 3.12, loc_{min} and loc_{max} refer to the minimum and maximum percentage of the virtual cluster to deploy in the selected providers.

$$loc_{min}(k) \leq \frac{\sum_{i,j}^{n,l} X_{i,j,k}}{n} \leq loc_{max}(k), \quad 1 \leq k \leq m \tag{3.12}$$

To understand better the placement constraint, here we mention some typical cases:

- Place at least 10 % of the infrastructure in each available cloud. Therefore, $loc_{min}(k) = 0,1 \;\; 1 \leq k \leq m$.
- Place no more than 50 % of the infrastructure in a single cloud. Therefore, $loc_{max}(k) = 0,5 \;\; 1 \leq k \leq m$.
- As Cloud X is the nearest to me, place between 40 % and 60 % of the infrastructure in Cloud X. Therefore, $loc_{min}(k) = 0,4$, $loc_{max}(k) = 0,6$.

Unity

To complete the model, we introduce the unity constraint, which ensures that each VM belongs only to one instance type, and it is placed in exactly one cloud provider.

- *Unity constraint*:

$$\sum_{j,k}^{l,m} X_{i,j,k} = 1, \quad for\ all\ VM_i, \quad 1 \leq i \leq n \tag{3.13}$$

3.3. Price forecasting

As introduced in cost optimization policy details on Section 3.2.3, in the case of dynamic scenarios we need to estimate the prices in different instance types of different clouds before the beginning of the scheduling period.

Therefore, in this section we formulate the different estimation methods used to forecast the prices of instances for those cloud providers that offer dynamic pricing schemes. The goal of our forecasting approach is to optimize cost by getting as close as possible to oracle prices, which are the best prices in each period (t), and which are unknown until the end of the scheduling period.

In this work, we have used the Amazon EC2 historical spot prices as input data to develop our forecasting methods. This prices are available using the AWS Management Console, and can be also obtained through the Amazon EC2 API, or using Amazon SDK. Each resource type in any placement has a different start price, and then it fluctuates on the basis of its particular demand. In other words, prices can increase, decrease, or be the same as last hour prices, even deployed in different regions offered by the same cloud provider.

Proposed forecasting algorithm

The goal here is to find an optimal resource deployment for next period t before knowing the prices of each cloud provider ($P^t_{j,k}$), so one valid solution is to try to predict next hour prices using estimations, so we define $E^t_{j,k}$, as the estimated price of a virtual machine of type it_j deployed in cloud k during period t.

In our first approach, the computation of $E^t_{j,k}$ is based on the two following parameters:

- The **average** price of a cloud provider, $\overline{P}_{j,k}$
- The **trend** of a cloud provider, $\tau_{j,k}$

The average is defined as follows:

$$\overline{P}^t_{j,k} = \frac{\sum_{t=1}^{n} P^t_{j,k}}{n} \tag{3.14}$$

In equation 3.14 we calculate the sum of last observed prices of a virtual machine in a particular cloud provider. And then, this sum is divided by the period of time used for this calculation, to result in an average value.

If we use $\overline{P}_{j,k}$ parameter over a time interval to adjust our function to minimize, we obtain a logical prediction about where to deploy our virtual resources. However, the scheduler does not know where, within the considered time interval, the cheapest prices appear (if near or far from the next

hour to predict). And obviously, recent price trend is interesting for making predictions.

As a result, we have defined the trend parameter $\tau_{j,k}$ as follow:

$$\tau_{j,k}^{t} = \begin{cases} 1,05 & \text{if } P_{j,k}^{t-1} > P_{j,k}^{t-2} \geq P_{j,k}^{t-3} \\ 0,95 & \text{if } P_{j,k}^{t-1} < P_{j,k}^{t-2} \leq P_{j,k}^{t-3} \\ 1 & otherwise \end{cases} \tag{3.15}$$

We define *Trend* as the relationship between three last observed prices. As the last known price can be higher, lower or the same as the previous one, we define three types of trends:

- increasing, when next price is supposed to get higher;
- decreasing, when next price is supposed to get lower;
- and constant, when next price is supposed to maintain the previous prices trend.

An increasing trend in a cloud provider means that this provider is holding a high load level in this interval. Similarly, a decreasing trend in a cloud provider means that it is not overloaded and its physical resources are up to receive more clients. The possible values for $\tau_{j,k}^{t}$ are the penalty that $E_{j,k}^{t}$ will suffer depending on the trend.

With these two parameters, $\overline{P}_{j,k}^{t}$ and $\tau_{j,k}^{t}$, we define the estimated cost as:

$$E_{j,k}^{t} = \overline{P}_{j,k}^{t} \cdot \tau_{j,k}^{t} \tag{3.16}$$

And substituting $E_{j,k}^{t}$ in equation 3.3 (introduced in Section 3.2.3), results in:

$$\begin{aligned} TIC_{est}^{t} &= \sum_{i,j,k}^{n,l,m} X_{i,j,k}^{t} \cdot E_{j,k}^{t} \\ &= \sum_{i,j,k}^{n,l,m} X_{i,j,k}^{t} \cdot \overline{P}_{j,k}^{t} \cdot \tau_{j,k}^{t} \end{aligned} \tag{3.17}$$

By minimizing equation (3.17) we address the challenge of deploying virtual resources in the clouds that we consider the cheapest.

Other forecasting algorithms

The objective algorithm aforementioned, which we design as our initial experimental proposal, takes three last observed prices for each instance type on each cloud provider.

In literature there are lots of works about forecasting methods, and we want to study deeper some of these forecasting algorithms from literature in order to compare our initial algorithm or definitely take the one that performs better. Therefore, to improve predictions over $E_{j,k}(t)$ values in the rest of this work, we consider several forecasting methods [CH04].

These forecasting methods are used to estimate future behaviour as a function of past data, so they are only appropriate when past data are available. However, we have not considered forecasting methods based on cyclical phenomena because spot prices do not follow any specific pattern, regarding hours of a day, and days in a week [JTB11].

Some example of quantitative forecasting methods are the following:

Last period data (LPD)

Last period data (LPD), which guesses correctly in cases where the cheapest price occasionally (or never) changes from one provider to another, failing its decision in each of these changes.

$$LPD^t = data^{t-1} \tag{3.18}$$

Simple moving average (SMA)

Simple Moving Average (SMA), which uses several past data to show the best average price in a single moment, but is not aware of prices trend. An increasing trend of a data set is not the same as a decreasing trend, although both data sets can have the same average value.

The theoretical definition of the SMA method is the following:

$$\begin{aligned} SMA^t &= \frac{data^{t-1} + data^{t-2} + ... + data^{t-n}}{n} \\ where\ n &= number\ of\ selected\ data \end{aligned} \tag{3.19}$$

When dynamic prices are produced by the variable demand, recent prices trend becomes interesting for making predictions. An increasing trend in a cloud provider means that this provider is holding a high load level in this interval. Similarly, a decreasing prices trend in a cloud provider means that it is not overloaded, and its physical resources are ready to receive more clients. Thus, we considered some trend-aware methods.

Weighted moving average (WMA)

Weighted Moving Average (WMA), which over-performs SMA allowing to value data which are closed to the scheduling period, by assigning different weights to these data.

This method is well-known in the stock market strategy. It can be used to see trends, predict if data is bucking the trend, and also to smooth out short-term fluctuations. It has multiplying factors to give different weights to data at different positions in the sample window, assigning a greater weight to the more recent data.

The theoretical definition of the WMA method is the following:

$$WMA^t = \frac{n \cdot data^{t-1} + (n-1) \cdot data^{t-2} + ... + 2 \cdot data^{t-n+2} + data^{t-n+1}}{n + (n-1) + (n-2) + ... + 2 + 1} \tag{3.20}$$

Exponential moving average (EMA)

Exponential Moving Average (EMA), which is quite similar to WMA but the weights assigned to data decrease in exponential progression, instead of arithmetical progression. This method is aware of the trend of data pools.

The theoretical definition of the EMA method is the following:

$$\begin{aligned} &EMA^t = n_1 \cdot data^{t-1} + n_2 \cdot data^{t-2} + ... + n_z \cdot data^{t-z} \\ &where \;\; n_i = \frac{(1-\alpha)^i}{1 + (1-\alpha) + (1-\alpha)^2 + ... + (1-\alpha)^i} \\ &and \;\; 1 > \alpha > 0 \end{aligned} \tag{3.21}$$

In Section 4.1.4, we show a comparison of these prediction methods applied to a real scenario.

3.4. Storage-aware brokering

In this section we present an extension of the aforementioned brokering algorithms. Hence we explore the convenience of using brokering mechanisms to reallocate part or the entire infrastructure to another cloud placement by taking into account not only the compute cost, but also the image storage cost.

Although this is a multi-cloud challenge, in the evaluation section, we only consider Amazon EC2 to simulate different clouds by using its regions as independent isolated clouds. Other cloud infrastructures can easily be added to the simulator, provided that accurate models of their costs and overall architecture are available.

As a short summary, Amazon offers the following types of storage:

- Glacier, which is a secure, durable, and extremely low-cost storage service for data archiving and on-line backup. It is optimized for infrequently accessed data where a retrieval time of several hours is suitable.

  For this reason, in this work don not consider Glacier as a valid type of storage.

- Elastic Block Storage (EBS), which is designed specifically for EC2 instances, and allows users to create remote block storage volumes that can be mounted as devices by EC2 instances.

- Simple Storage Service (S3), which provides a simple web service interface that can be used to store and retrieve any amount of data, at any time, from anywhere on the web. It is billed in GB of data per month, and there are billing intervals in which Amazon reduces cost per GB as more data is consumed.

Due to annual failure estimations, EBS users should keep an up-to-date snapshot on S3, or have a backup of the contents somewhere else that they can restore quickly enough to meet their needs in the case of a failure.

On the other hand, S3 is subject to eventual consistency, which means that there may be a delay in writes appearing in the system whereas EBS has no consistency delays. Also EBS can only be accessed by one machine at a time whereas snapshots on S3 can be shared between different VMs. EBS volumes can only be accessed from an EC2 instance in the same availability zone whereas snapshots on S3 can be accessed from any availability zone in one region.

Proposal

Our proposal consists on adding storage price information to the decision algorithm process, and validate our algorithms on SimGrid Cloud Broker (SGCB). As SGCB is coded in Java, we select Choco Constraint Programming Library [JRL08] to develop the algorithm within the simulator.

For this proposal we consider the same definitions as explained at the end of Section 3.2.2 introduction. In short, our goal is to deploy a number n of VMs, $v1, ..., vn$, that belong to certain instance types it_j, $j_1, ..., j_l$, across the m available clouds, $c1, ..., cm$ to optimize user criteria such as cost or performance.

Concretely, here we want to minimize the Total Cost of the Infrastructure with Storage (TICS), which is formulated as follows:

$$TICS^t(x) = \sum_{i}^{n} \sum_{j}^{l} \sum_{k}^{m} X_{i,j,k}^t \cdot C_{j,k}^t(x) \tag{3.22}$$

with

$$C_{j,k}^t(x) = P_{j,k}^t \cdot SC_k(x) \tag{3.23}$$

where:

- $P_{j,k}$ refers to the price of an instance type j in a cloud k under Amazon's SI pricing scheme.

- and $SC_{x,k}$ refers to the price of storing x bytes in cloud k.

Finally, for the experimentation discussed in Section 4.3, we use the same objective functions and brokering constraints presented in Sections 3.2.3 and 3.2.4.

Storage policies

We propose three different policies that consider Virtual Machine Image (VMI) allocation within cloud brokering: VMI uploading, deletion, and transfer. Here we introduce all of them:

- Uploading policy.

  Before deploying an instance type in a cloud provider, we need to indicate which VMI we want to boot. Considering storage costs and the features that a VMI provides, there are two options:

  On one hand, cloud providers generally offer some VMI for free, but they consist on basic VMIs with an operating system (Linux or Windows in general) and without specific packages that can be useful for users needs.

  On the other hand, the user can upload a customized VMI, or customize one of the offered proprietary VMI, but at a certain cost per period of time (generally per month).

  In our storage experiments we consider the necessity of always using a customized VMI, so the decision of when to upload the VMI is critical for our optimization purposes.

- Deletion policy.

  Once an instance is terminated in a cloud provider, the user has to decide what to do with the VMIs that were uploaded before. If we take into account the next scheduling period, logically we must delete the VMI in order to optimize costs. But if we take into account the whole infrastructure life cycle, the previous consideration is not always true, since if we maintain the VMI in the public cloud, we do not have to upload again in case that we need.

  In our storage experiments we always consider dynamic scenarios, in which the infrastructure can change its placement along time, so the decision of when to delete the VMI is also a parameter to consider.

- Transfer policy.

  During the infrastructure life cycle, the broker can reallocate part of it to other clouds as explained in Section 3.2.4. And, as introduced in the

uploading policy, we need a VMI in the destination cloud if we want to deploy a VM.

Apart from uploading directly the VMI from users placement to the correspondent cloud, some cloud providers offers a way to transfer the images from one region to another one, which usually is faster and cheaper for users, since cloud providers use their own internet physical network connections and they offer it as value-added for their users.

In our storage experiments we consider Amazon EC2 regions as different clouds, and Amazon offers the aforementioned possibility, so the decision of transfer the VMI from users placement or from the last used region to the new one, is as critical for our optimization purposes as the other policies.

Once explained these policies, we need to study deeper each of them. Therefore, for the **uploading policy**, we propose two strategies.

- The first one is called `Everywhere` (E) and it specifies that the VMI must be uploaded into all the potentially used clouds at the beginning. Accordingly, even if a cloud is not used but is considered by the cloud brokering algorithm, the VMI will be uploaded there. This strategy has a monetary cost as the VMI is stored in all clouds during the whole life-time of the application.

- The second uploading strategy is called `On-Demand` (O) and it specifies that the VMI is uploaded to a cloud only when the brokering algorithm has specified that at least one VM will be started there. Accordingly when a cloud is not used, the VMI is not stored in it. But this approach requires to upload the VMI to a cloud before being able to start a VM there. Therefore it can induce delay on the VM startup, *i.e.* it adds the uploading time of the VMI.

Next, for the **deletion policy**, we also propose two strategies.

- The first one is called `Never` (N) and it specifies that once a VMI is uploaded to a cloud it will never be deleted, *i.e.* until the end of the application's life. Accordingly, even if no VM are running in a cloud, the VMI is still stored on it. Therefore, this strategy has a monetary cost as the VMI is stored on a cloud even if it is not used.

- The second deletion strategy is called `Always` (A), and it specifies that when there is no VMs running on a cloud, the VMI must be deleted there. Accordingly, this strategy allows to reduce the monetary cost by only storing the VMI where it is needed. But, as the `O` strategy for uploading the VMI, it can induce a delay on the VM startup as it could be required to re-upload several times a VMI to the same cloud.

And finally, for the VMI **transfer policy**, we evaluate two different strategies to transfer the VMI to a cloud.

- The first one (`Get`) specifies that the VMI is uploaded by the user to the cloud.
- The second one (`Copy`) specifies that the VMI is uploaded once by the user to the first cloud and then the VMI is copied from a cloud to another one. The second strategy must able transfer the VMI faster as network links between clouds are faster than the ones between the user and each cloud. But, in AWS at least, uploading data from a user to a region is free, but copying between region has a cost.

Chapter 4

Experiments and results

Una experiencia nunca es un fracaso,
pues siempre viene a demostrar algo.
Thomas Alva Edison

In this Chapter we introduce and explain the experiments performed during this thesis, considering the proposals exposed in Chapter 3, and drawing conclusions from them. The aim of these experiments is to probe the benefits of the brokering algorithms to simulated scenarios designed to be similar to real world ones. We perform these experiments considering the two brokering scenarios explained: static and dynamic.

The outline of this Chapter is as follows:

- First we explain a preliminary stage (Section 4.1) to develop, verify, and tune-up our simulator. In this stage, we test basic scenarios (static ones), we do an early analysis of complex scenarios (dynamic ones), and a first approach to price forecasting methods. As a result, in this stage we set the bases of this work, that we later apply to complex real scenarios.

- Later, considering complex scenarios, we focus on real world use cases deployments, applying the scheduling algorithms to generic clustered applications (Section 4.2). This section outlines three different real use cases, such as generic clusters, HPC cluster, and Web Servers. With this section we give the reader an overview of how brokering mechanisms can be applied to not only academical use cases. Moreover, in this stage we work in modelling and studying real performance metrics for each aforementioned case.

- Finally we conclude this Chapter introducing our storage-aware brokering proposal (Section 4.3). Storage is an important component to

take into account when interacting with IaaS public cloud providers, since VMs boot from images stored in these clouds. However, few works consider this as a key point. In the experiments we demonstrate that cloud users reduce their investment in infrastructure if they take into account different storage policies.

To perform the experiments of the first and second stages we use AMPL, as introduced in Section 3.2.3. For the storage experiments, we use the SimGrid Cloud Broker (SGCB) simulator [DRC13]. In all cases, we use the full EC2 platform with all regions -see equation (3.12)- and instance types -see equation (3.11)-. Within SGCB, we use its random spot instance price statistical distribution, saving the data set for experiment reproduction. All other non-spot prices have been retrieved from the AWS website [1].

4.1. Preliminary results

The goal of the experiments of this section is to adjust the proposed mathematical formulation for objective functions and constraints, as well as the price forecasting algorithms. As commented in Section 3.2.1, the cloud scheduling challenge can be addressed using either a static or a dynamic approach.

The static approach is suitable for situations where the number of required virtual resources does not change (for example, a fixed-size service), and the cloud provider conditions remain unchanged throughout the service life-cycle (resource prices, resource availability, etc.). In this scenario the resource selection can be done off-line, once, and in advance to service deployment.

The dynamic approach is more suitable for variable size services (e.g. a web server with fluctuating resource requirements), or in the case of changing cloud provider conditions (variable prices, or dynamic resource availability). In this case, the optimization algorithm runs periodically to adapt the current infrastructure to the variable resource requirements and cloud conditions.

4.1.1. Static scheduling

The aim of this experiment is to test the behaviour of the cloud broker for static deployments. The goal is to optimize the overall infrastructure cost for an user, considering only the on-demand pricing scheme (not spot neither reserved pricing schemes), static prices, and pre-defined (standard) or user-made instance types. This experiment reproduces the typical cloud-comparison challenge of a cloud user when trying to deploy a simple infras-

[1] For EC2 `http://aws.amazon.com/ec2/pricing/` and for S3 `http://aws.amazon.com/s3/pricing/`

tructure in a single cloud, when different clouds with different offerings are available.

For this experiment, we use the cost optimization strategy (see equation 3.1), and we consider the following constraints:

- Placement, since we select three cloud providers (Amazon EC2 [2], ElasticHosts [3], and OpSource [4]) to work with.
- Instance type, since we use a mixture between pre-defined instances offered by almost every cloud provider (such as small, large, and xlarge instances), and customized instances, that can be defined by the user according to their needs of CPU, RAM, and disk (HDD).

Table 4.1 shows the features of three standard instance types in terms of hardware components. Pre-defined instances are offered at a special fixed price by certain providers, such as Amazon EC2 among others. Another providers, such as OpSource or ElasticHosts, offer these instances in a price-per-component pricing scheme, in which users can customize their virtual machines (Amazon EC2 does not allow users to customize their own virtual machines). Moreover, it shows the prices-per-component at the right side, whereas the cost of the entire instance type in considered cloud providers (except for extra large instance in ElasticHosts, where users can not reserve more than 8 GB of RAM) is at the bottom of the table.

	Standard configurations			**Resource prices €/h**	
	Small	**Large**	**XLarge**	**OpSource**	**ElasticHosts**
CPU (ECU)	1	4	8	0,04	0,018
RAM (Gb)	1,7	7,5	15	0,025	0,025
Storage (Gb)	160	850	1690	0,0003	0,0014*
	Instance prices €/h				
EC2	0,095	0,380	0,760		
OpSource	0,130	0,602	1,202		
ElasticHosts	0,284	1,449	—		

Table 4.1: Instance types features and prices (2012).
* Real price: 0,10 € /Gb-month, which is equivalent to 0,0014 € /Gb-hour

We observe at first sight that Amazon presents cheaper prices than the other two providers, but in this provider there is no possibility of using customized instances, but only pre-defined ones. Moreover, in Table 4.1 we observed that OpSource offers the cheapest hard disk storage whereas ElasticHosts offers cheapest cores, or that the cheapest provider is Amazon EC2 when the resources required fit the pre-defined instance type.

[2] Amazon Elastic Compute Cloud (EC2) - `http://aws.amazon.com/ec2/`
[3] ElasticHosts - `http://www.elastichosts.com/`
[4] Op Source, a Dimension Data Company - `http://www.opsource.net/`

In the experimental part we evaluate two experiments, in which the infrastructure to deploy is expressed in terms of number of cores, or amount of RAM or disk. The combinations of these resources can fit into one or more VMs, depending on the instance types considered. These experiments are the following:

1. In the experiment 1, we consider that the CPU and RAM requirements fit in a pre-defined instance type, but the minimum capacity of disk required is variable.

2. In the experiment 2, we consider that the user requirements do not fit in any pre-defined instance type features.

We split the first experiment into two different cases, and we show in Table 4.2 the constraints used in both cases.

Table 4.2: Constraints used in experiments 1 and 2.

	Deploy. Type	**Constraints**	
		Performance	**I.Type**
Exp. 1(a)	Static	1 core , 1,7 Gb RAM, [10-80] Gb HDD	-
Exp. 1(b)		4 cores, 7,5 Gb RAM, [50-850]Gb HDD	
Exp. 2	Static	1 core, 1 Gb RAM, 50Gb HDD ... 4 cores, 4 Gb RAM, 50Gb HDD	

* Cloud placement constraint: using EC2 or OP or EH; Unity constraint always used.

In the experiment 1(a) (Figure 4.1), the user needs exactly 1 CPU and 1,7 Gb of RAM (similar to the *Small* instance features), and the minimum amount of HDD storage is variable. Figure 4.1 shows that, with less than 25 Gb of HDD the cheapest cloud provider is ElasticHosts; between 25 and 42 Gb of HDD, OpSource offers the cheapest solution; but for higher amount of HDD, the cheapest solution is to reserve the *Small* pre-defined instance in Amazon EC2 because it is cheaper and it provides more storage capacity than the others.

In the experiment 1(b) (Fig 4.2), the user needs exactly 4 CPU and 7,5 Gb of RAM (similar to the *Large* instance features), and the minimum amount of HDD storage is variable. Here, with less than 86 Gb of HDD, the cheapest cloud provider is ElasticHosts; OpSource offers the cheapest solution between 86 and 108 Gb of HDD; and for higher amount of HDD, the cheapest solution is to reserve the *Large* pre-defined instance in Amazon EC2 because it is cheaper than the others and it provides users 750 Gb of HDD.

In experiment 2 (see also Table 4.2), the user requirements do not fit in the pre-defined instance type features. In this case, the user needs the same amount of CPU cores than Gb of RAM, e.g. 1 core and 1 Gb of RAM, 2

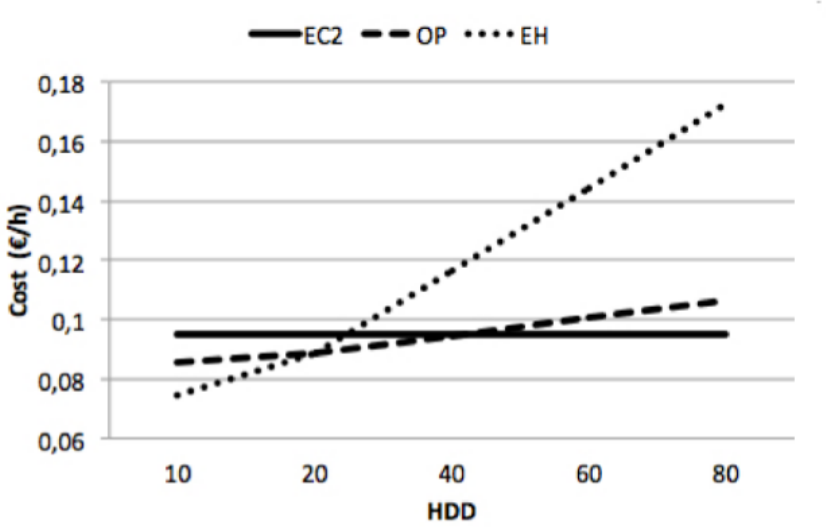


Figure 4.1: Experiment 1a: optimal cost, small instance, different HDD requirements.

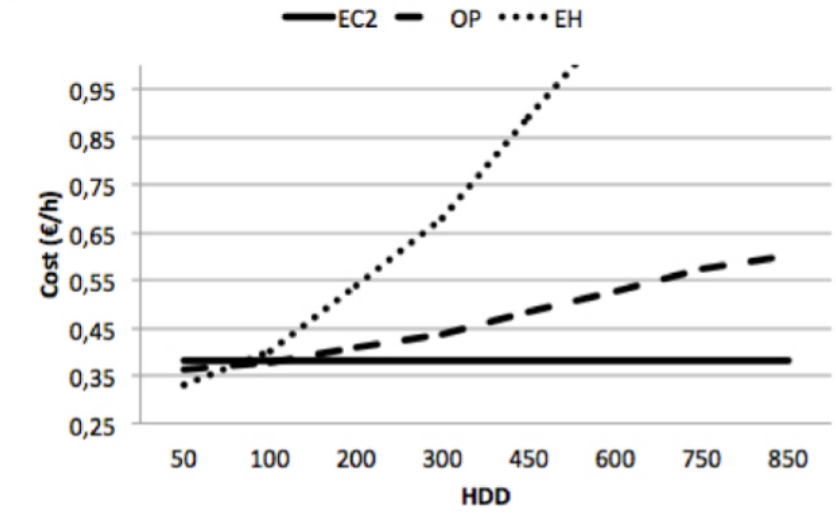


Figure 4.2: Experiment 1b: optimal cost, large instance, different HDD requirements.

cores and 2 Gb of RAM and so on, but with a fixed amount of 50 Gb of HDD.

Hence, in Amazon, the user has to reserve one small instance for each core needed, while in the other providers the user can reserve one single virtual machine customized accordingly to the set of requirements. In this experiment, we execute the broker in order to deploy optimally the infrastructure required, considering providers that offer customizable VMs, and we compare the resulting TIC to the cost of a supposed Amazon deployment.

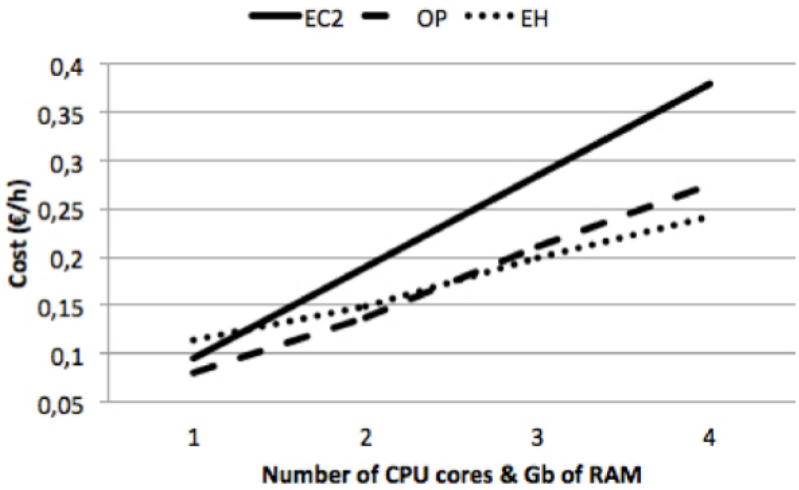


Figure 4.3: Experiment 2: Optimal cost with different combinations of CPU cores and RAM.

Figure 4.3 depicts the cost of each deployment. Regarding 1 and 2 cores experiments (same amount of Gb of RAM), OpSource provides the best solution, whereas regarding 3 and 4 cores experiments, ElasticHosts provides the optimal solution. Notice that Amazon over-performs ElasticHosts in the 1-core case.

Figure 4.4 depicts the cost comparison (normalized against the cheapest result) for each case. This is to say that in the single-core experiment, the user is paying 20 % more if he/she chooses Amazon instead of OpSource, or over 40 % more if he/she chooses ElasticHosts. However, in the 4 cores experiment, ElasticHosts offers the required resources with a saving of 14 % if compared to OpSource, or 52 % if compared to Amazon.

To finish this experiment we conclude that, using the broker for static deployments, users can find an optimal deployment for exactly their needed requirements, avoiding to pay for unused resources. The percentage of savings is not the goal of this experiments, but it is the way users can take advantage of cheapest clouds in every moment.

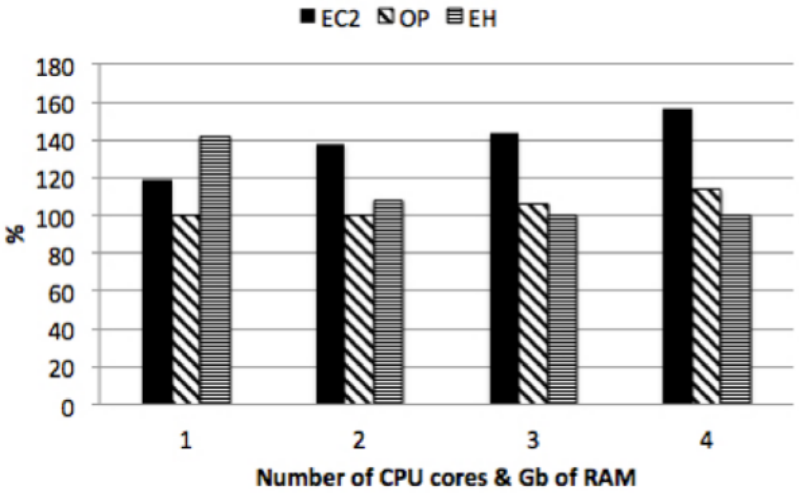


Figure 4.4: Experiment 2: Cost comparison.

Considering these results, we also realize that the possibility of changing the cloud provider dynamically within the same deployment could improve savings and benefits.

4.1.2. Dynamic scheduling potential

We consider a dynamic deployment when it is done periodically, regarding the available conditions of the cloud market in each moment of the scheduling period. For dynamic scenarios we evaluate in our experiments the impact of two parameters explained in Chapter 3: *spot pricing scheme*, and *oracle prices*.

In this experiment, and in almost every experiment from now on, we extract the experimental data by consulting Amazon EC2 historical spot prices. As spot-prices are a key point in this work, here is a short explanation of them:

Generally, each resource type in any placement has a different base price, and then it fluctuates based on its particular demand. In other words, prices can increase, decrease or maintain similar as last hour prices, even deployed in different regions offered by the same cloud provider.

Spot-pricing rules permit providers to terminate users resources if bid price is lower than the spot price. However, we have not taken this feature into account in any experiment of our whole work, since it is very unlikely that the instances will be terminated if users choose a very high bid price. We decided to use spot prices variations to simulate a realistic dynamic pricing cloud environment.

In Figure 4.5, we show the trend of Amazon EC2 historical spot prices from a 24 hours period. These prices belong to a *small* instance type in three different regions: United States West (USW), Asia (AS) and Europe (EU).

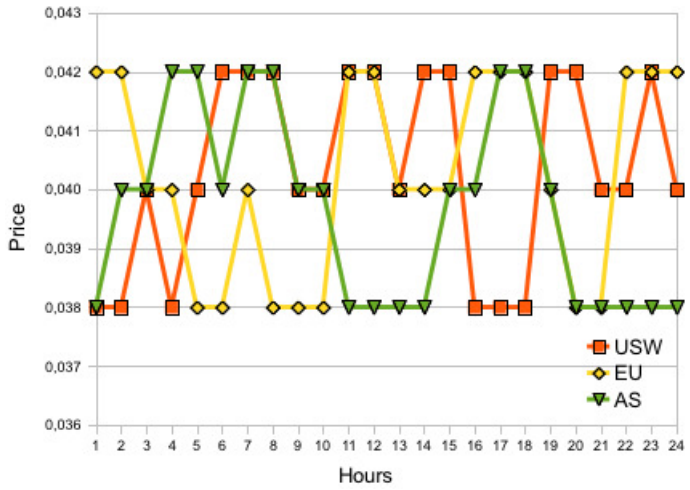


Figure 4.5: Hourly Prices.

Oracle prices are also crucial to understand most analysis and comparisons in this work. Oracle prices mean the best real prices of each instance type in each scheduling period t, and the oracle deployment means the cheapest possible deployment considering all available instance types and clouds.

The goal of this experiment is to compare the dynamic scheduling potential against static deployments. This experiment reproduces the challenge for cloud users of changing their infrastructure placement in an easy and automated way, and get economical benefits. Hence, static scheduling consists on hold the whole deployment (in a single cloud or multiple clouds) during the scheduling period (24h). In dynamic scheduling, the broker can move the VMs in each scheduling period (hourly) considering Oracle prices. So, we use the broker with our cost optimization strategy (see equation 3.1), and also we consider the following constraints:

- Placement, since we select three regions of the same cloud provider (Amazon EC2) to work with, considering them as different clouds, and only using one of them at the same time.
- Instance type, since we set a fixed instance type to work with: small instance type in this case.

We have simulated a virtual infrastructure composed by 10 VMs during a 24 hours period, considering the aforementioned scenarios:

- Static, in which the experiment consists on maintaining the whole set of virtual machines in the same placement during the 24 hour period. We divide this case into:

  - *Single cloud deployment*, where the entire cluster is deployed in one cloud region (as we work with three regions -USW, EU and AS-, we measured this cost in each one).
  - *Balanced deployment*, where VMs are equitably distributed among different cloud regions.

- Dynamic, which consists on moving the whole cluster to the cheapest cloud region in each period t assuming that we know a priori the real prices of each time period (*oracle deployment*), and having the VMs initially placed at USW region because of its cheapest price in the selected data set (Fig. 4.5).

Figure 4.6 shows the difference between both types of deployment. Comparing the static deployments, the balanced one is cheaper than two of the single ones, so if we could not make any placement change and we don't know future prices, this deployment ensure a good choice. Moreover, our interest is to know the improvement potential of the dynamic scheduling by comparing the oracle deployment with the static ones.

In this scenario, in which price difference between the cheapest cloud provider and the most expensive one is not very high (see prices in Figure 4.5), we observe that users can save up to 5 % per day of their investment using dynamic deployment. In environments with a higher price variability, the economical benefit can be higher.

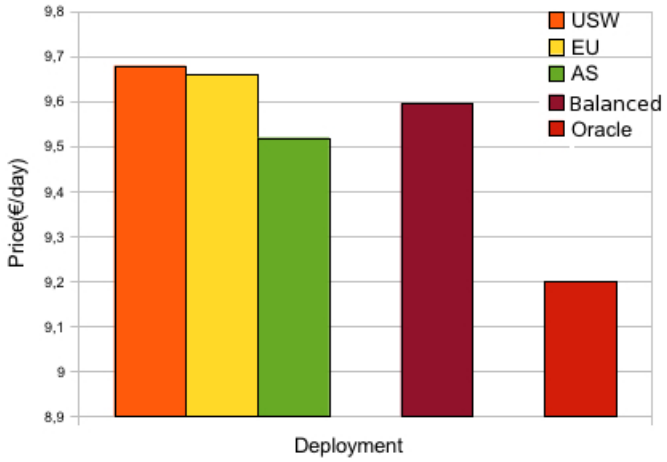


Figure 4.6: Static vs Oracle Deployment.

4.1.3. Initial price forecasting method

Looking at results shown in Figure 4.6 about economical benefits in an ideal dynamic scheduling, we realize that forecasting prices is a good way to optimize deployment costs in a dynamic scenario. However, in this scenario, we want to introduce the typical issue when moving infrastructure from one cloud to another. When part of the infrastructure changes its placement (always in dynamic cases), the performance degradation problem appears during the short period in which some VMs are being shut-down and then started in another placement. In general, it takes few seconds -even minutes-, and it only affects to the part of virtual infrastructure reconfigured. In the system formulation we define the reallocation restriction (equation 3.10) together with the function to optimize (equation 3.17) for managing these situations.

The aim of this experiment is to evaluate the price forecasting method proposed in Section 3.3, and the integration of these estimations with the broker, by comparing different dynamic deployments using estimated prices and different allocation constraints with the static and the oracle deployments.

In the experiments of this section we evaluate the impact of another two parameters: *price forecasting* and *reallocation constraint*. And within the experimental part, we evaluate two experiments:

1. the first one, focused in the evaluation of the first price estimation model proposed in Section 3.3.

2. the second one, focused on the evaluation of the reallocation constraint, designed to avoid infrastructure performance degradation.

For making good-enough predictions, estimated prices are expected to approximate to the real (oracle) ones. In our proposal (see Section 3.3), estimated prices are calculated using equation 3.16. Therefore, the goal of the first experiment of this section is to test how our proposed forecasting algorithm performs when we apply it to the data set retrieved for Section 4.1.2 experiments. In Figure 4.7 is depicted the estimated prices for USW region, compared to the real ones. We observe that the scheduler predicts prices and trends in a satisfactory way.

The goal of the second experiment is to test the broker for optimizing the cost of an infrastructure in a dynamic scenario, with multiple clouds, using the price forecasting method. In addition, we also use the reallocation constraint to avoid performance degradation.

In this second experiment we use the broker with our cost optimization strategy (see equation 3.1), and also we consider the following constraints:

- Placement, since we select same three regions Amazon EC2 considering them as different clouds, and only using one of them at the same time.

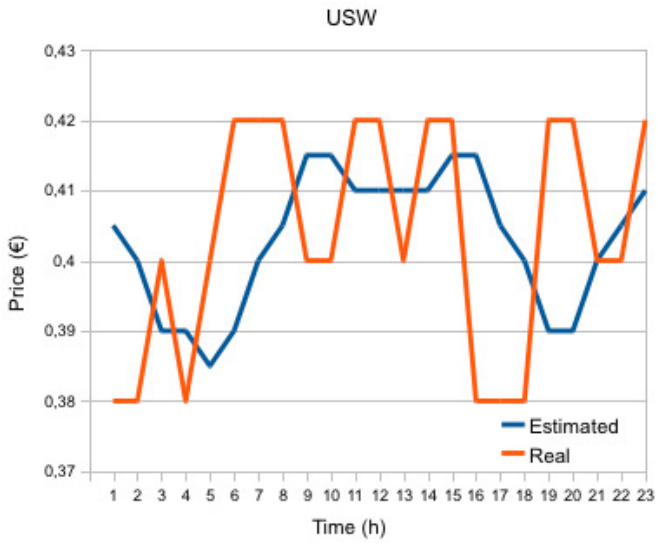


Figure 4.7: Experiment 1: Price estimation applied to USW Data

- Instance type, since we set a fixed instance type to work with: small instance type in this case.

- Reallocation, since we experiment with different reallocation percentages: up to 10 %, 20 %, 30 %, 40 %, and 100 % of the infrastructure.

Figures 4.8, 4.9, and 4.10 represent the VM's hourly distribution among available clouds.

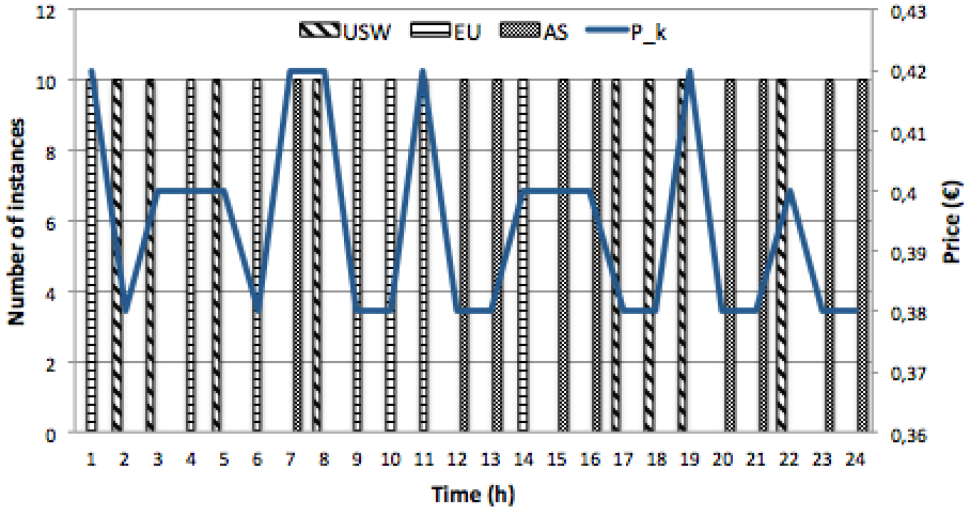


Figure 4.8: Reallocation constraint: up to 100 % infrastructure reallocated.

Figure 4.8 shows the predicted deployment with no reallocation restriction applied. Hence, the cloud broker can move all the VMs of the virtual infrastructure in each hourly scheduling decision.

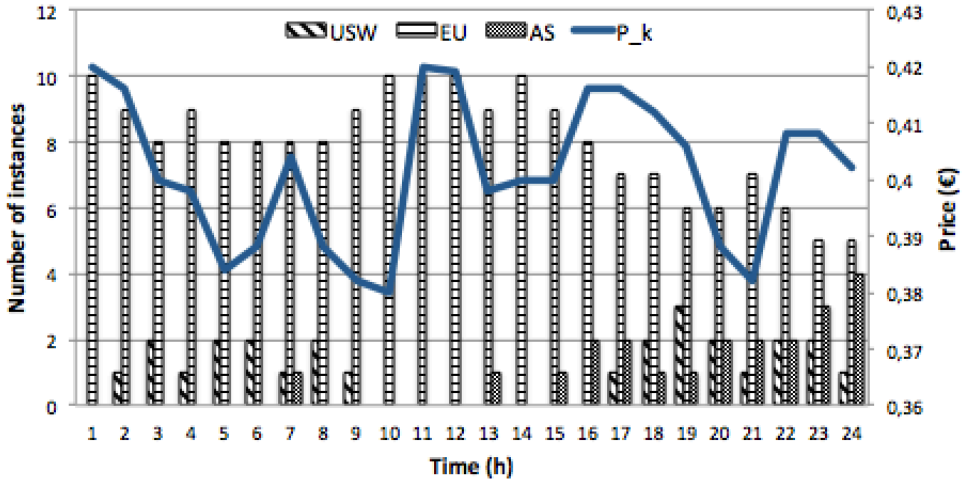


Figure 4.9: Reallocation constraint: up to 10 % infrastructure reallocated.

Figures 4.9 and 4.10 show the predicted deployments when users could afford the temporary performance degradation of 10 % and 40 % of the cluster with the benefit of decreasing their investment. Hence, only a maximum of 10 % and 40 % of the virtual infrastructure can be reconfigured respectively.

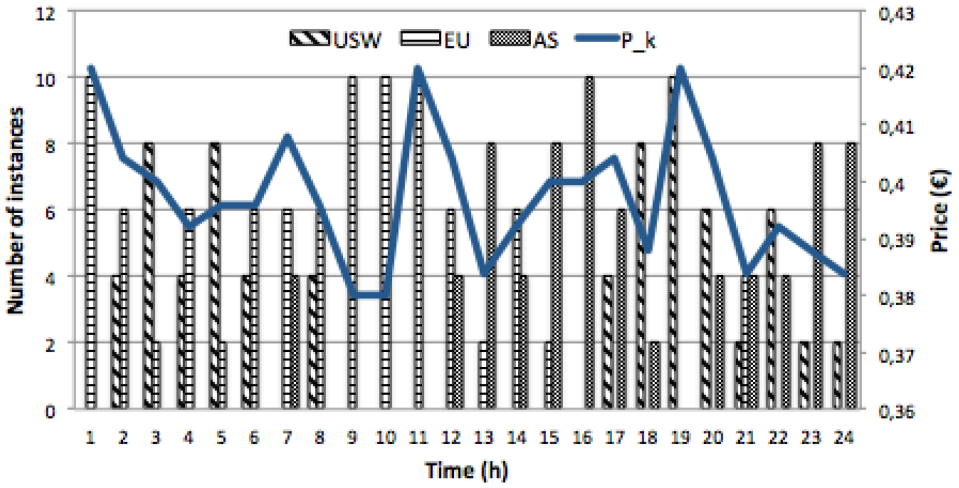


Figure 4.10: Reallocation constraint: up to 40 % infrastructure reallocated.

A real-world example of this behaviour is a Web server cluster divided into some VMs working as back-ends which are not used at maximum regime.

In this case, the Web server manager can take economical advantage by moving some VMs to another cheaper placement. Moreover, in Figures 4.8, 4.9, and 4.10 price evolution is indicated in solid line. Cost decreases in cases in which high part of the virtual infrastructure can be reconfigured. Nevertheless, all dynamic deployments show better performance than any static one.

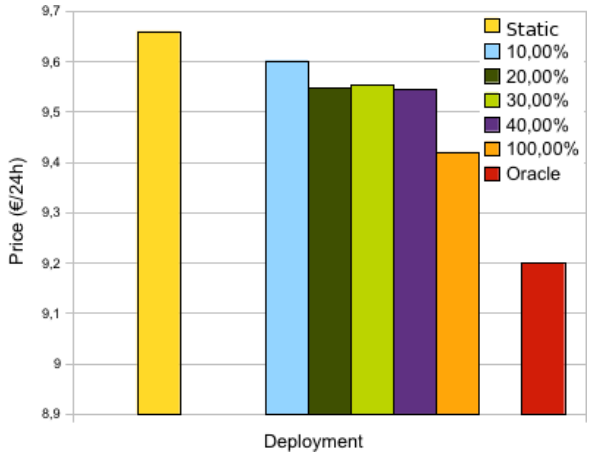


Figure 4.11: Cost comparison: Static, dynamic and oracle deployments.

Figure 4.11 shows the total cost of dynamic deployments compared with the static and the oracle ones.

And finally, Table 4.3 summarizes these results in a numerical way, and shows the improvement reached using dynamic deployment in terms of cost savings.

Table 4.3: Improvement of Dynamic Scheduling.

Improvement potential						
Static	**10 %**	**20 %**	**30 %**	**40 %**	**100 %**	**Oracle**
9,68	9,602	9,548	9,554	9,544	9,42	9,20
Saving	0,83 %	1,36 %	1,30 %	1,40 %	2,69 %	4,96 %

In this case, we compare a static deployment (using the EU region) with the oracle one, and some dynamic deployments with different distance restrictions. The oracle deployment is the cheapest one, so it is the target price to achieve.

Results show that dynamic deployments improve the static deployment price. So, as a conclusion of this forecasting section, the more part of virtual cluster can be reconfigured, the more benefit users get.

4.1.4. Other price forecasting methods

In Section 4.1.3 we use our own forecasting method, based on a combination of the simple moving average and trend of an instance type range of prices. For both parameters, we consider using few recent data per cloud, because using the whole data set results in similar averages, so it does not help us to predict correctly, and to know the cheapest cloud. For example, small instance type averages in Europe and USW regions tend to be similar if we use long data set.

In this section, our goal is to improve the forecasting algorithm to use from now on. Therefore, we need to check the quality of our proposed method against the different prediction methods explained in Section 3.3. For that purpose, we execute dynamically the scheduler, switching the prediction methods in each execution, and applying them to the same data set. To recapitulate, we consider our proposed method (ALG in Figures 4.12 and 4.13) and the following forecasting methods:

- Last Period Data (LPD).
- Simple Moving Average (SMA).
- Exponential Moving Average (EMA).

In this experiment we use the same data set from Section 4.1.2. We take into account on prices of a small instance type in the aforementioned three regions within Amazon EC2. Then, we compare the scheduler decision to the current best placement (here we use oracle prices), to check the number of successful predictions. This means the number of times each method predicts correctly the cheapest cloud provider for the next scheduling period.

Figure 4.12 shows a comparison between successful choices from these prediction methods in a one-day period. In this figure, the EMA method performs better than the other methods. As a result, we decide to weight recent data with higher multipliers to give more importance to data closed to the next scheduling period.

In addition to Figure 4.12, we compare the number of failed predictions (Figure 4.13), which means the number of times the method has chosen the most expensive cloud provider instead of the cheapest one. Here again, the EMA method shows the best behaviour, which corroborates the quality of this method. Therefore, we use the EMA method in the rest of this work to calculate the estimated TIC in dynamic scenarios experiments.

4.2. Use cases deployment

In this section, we apply the proposed performance optimization objective algorithm and performance constraint for the first time in this work. This

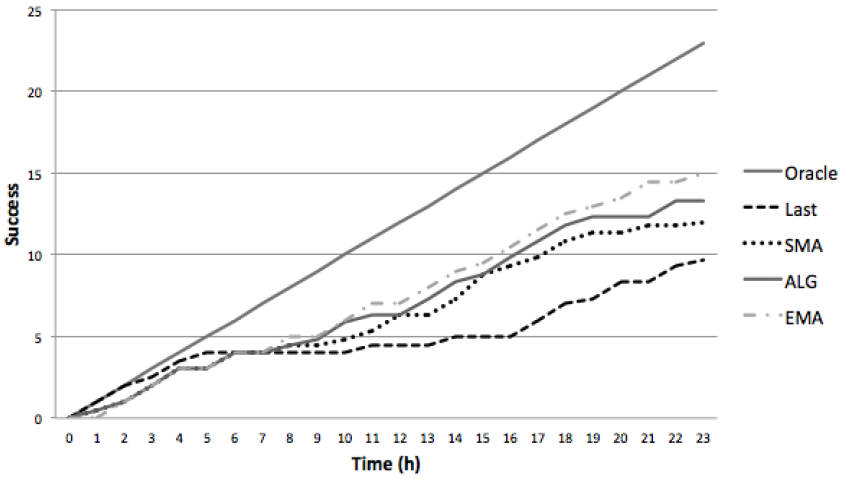


Figure 4.12: Forecasting comparison: successful choices.

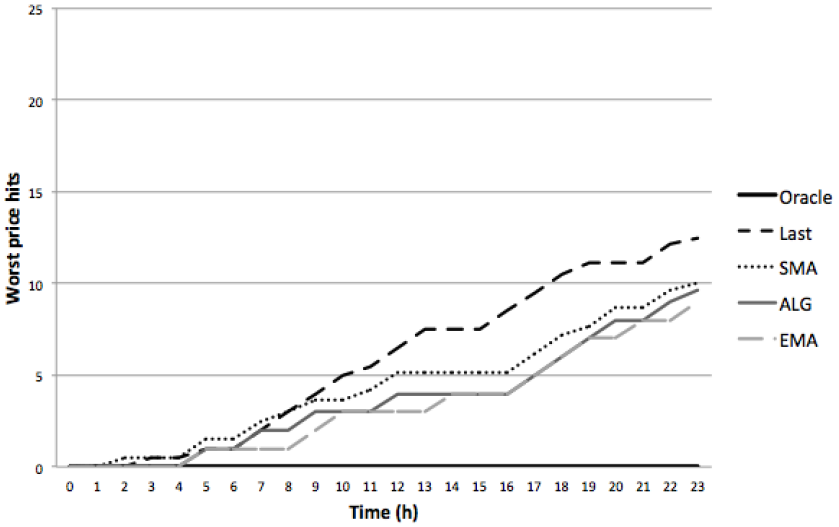


Figure 4.13: Forecasting comparison: wrong choices.

improvement, combined with the application of the rest of algorithms and constraints, makes our proposal more reliable.

To work with performance parameters it is necessary to define and analyse which metrics we consider in this work. Although there are other works in which the performance of Amazon EC2 instance types are analysed deeply, such as S.Ostermann et al. in [OIY+10], we decided to model each application in order to feed our broker with our performance input. Therefore, for each experiment we include a performance modelling analysis.

For the experimental part, we consider three use cases to deploy, going from generic to specific ones, specifically within performance modelling:

- Generic clusters, where generic patterns of industry clustered application can be included.

  We divide this case in homogeneous clusters, which are composed only by VM of the same instance type, and heterogeneous clusters, which can be composed by any type of instance available. In both cases we consider *number of cores* as our performance metric.

- High Performance Computing (HPC) clusters, which are one of the most deployed applications for research purposes (in academia as well as in private companies). In this cases we consider *MFLOPS* as our performance metric.

- Web Servers, which are one of the most deployed application in industry. In this cases we consider *request per second* as our performance metric.

From a cloud user point of view, the benefit of this Section is to deploy a service dynamically among multiple clouds, in an automated way, and focusing on performance metrics.

4.2.1. Generic cluster

The goal of this experiment is to evaluate the cost optimization function for dynamic multi-cloud scenarios in a real use case (a generic cluster) using infrastructure level performance constraints (number of cores), as well as different reallocation and instance type constraints.

Within generic clusters, we consider two use cases: homogeneous and heterogeneous clusters. The goal of both experiments is to optimize deployments costs (we still use our cost optimization strategy - see equation 3.1), to study how the broker moves VMs among different clouds, and to analyse the cost improvement potential of deploying VMs dynamically among multiple clouds, instead of maintaining them in a single cloud.

a) Homogeneous cluster

Within the *Homogeneous cluster* experimental environment, the performance constraint is expressed as the minimum number of virtual machines to reach. We define the following requirements:

- A cluster composed of 20 VMs.
- A single type of instance (*Small*).
- A experiment duration of 24 hours.

Table 4.4 resumes the constraints applied to our objective function in each case of homogeneous cluster experiments.

Table 4.4: Constraints used in homogeneous cluster experiments.

	Deploy. Type	**TIC**	**Constraints**			
			Perf.	**Reallocation**	**I.Type**	**Clouds**
Fig.4.14a	Dynamic	Oracle	20 VMs	0-100 %	Small	USW, EU, and AS
Fig.4.14b		Estimated		0-100 %		
Fig.4.15a		Estimated		0-10 %		
Fig.4.15b		Estimated		0-25 %		
Fig.4.15c		Estimated		0-50 %		

Figure 4.14 depicts the simulation results of deployments without reallocation restriction. Figure 4.14a represents the optimal VMs hourly distribution using the broker with the Oracle function, reallocating the whole infrastructure if needed. In the same way, Figure 4.14b shows the predicted deployment using the estimated TIC function, also without limits to the reallocation action. Therefore, both graphics depict how the broker moves the entire infrastructure to the best placement regarding its scheduling decision.

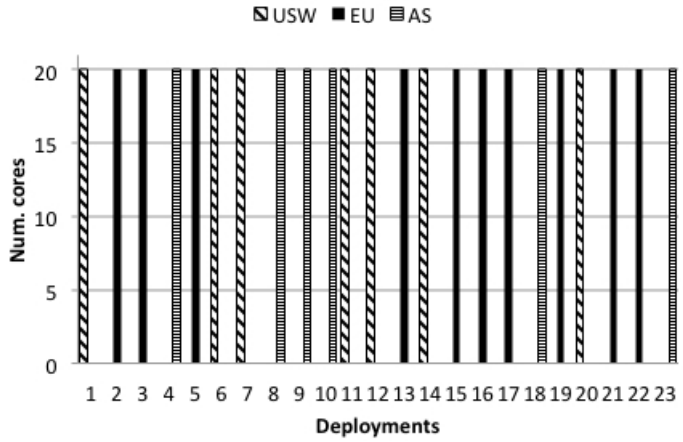
Figures 4.15a, 4.15b, and 4.15c show the optimal deployments with different reallocation constraints (10 %, 25 %, and 50 %), so that a maximum of 10 %, 25 %, and 50 % of the virtual infrastructure can be reconfigured.

Figure 4.16a, it shows the total cost of the dynamic deployments compared to the static and the oracle ones.

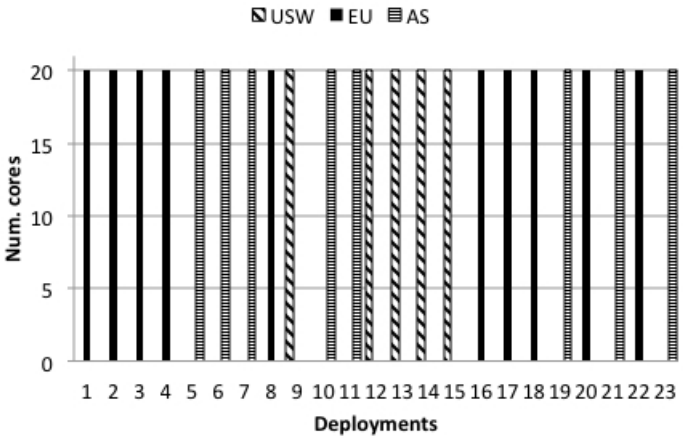
In summary, maintaining the VMs in a static placement is more expensive than using multiple clouds to deploy them. Regarding dynamic deployments, the bigger part of the virtual cluster can be reconfigured, the more benefit users get. And finally, the Oracle deployment is the optimal deployment and the objective to reach.

In addition, Table 4.5 summarizes these results numerically, and it shows the improvement we reach by using dynamic deployment in terms of cost savings.

b) Heterogeneous cluster



(a) Oracle TIC deployment

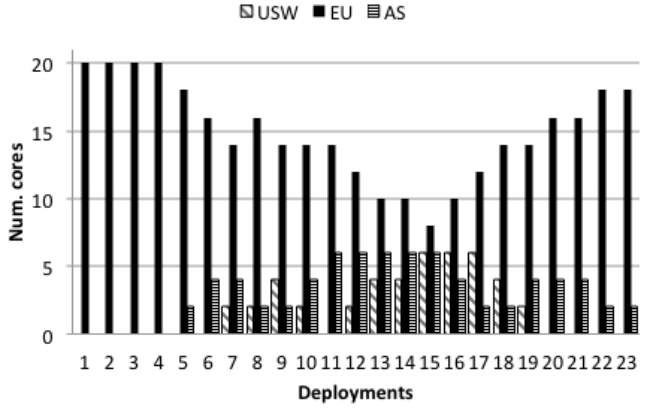


(b) Estimated TIC deployment.

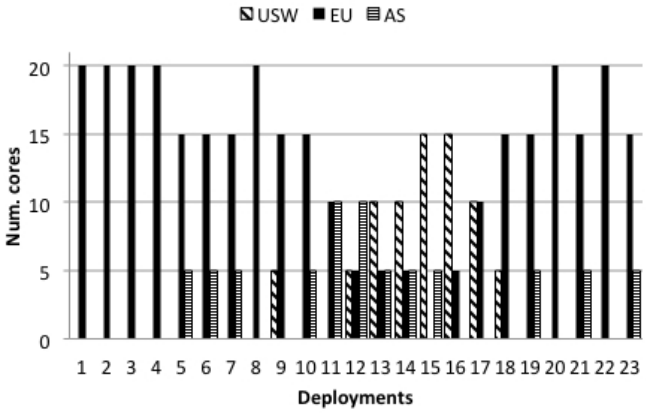
Figure 4.14: Cost optimization deployments without reallocation constraint.

Table 4.5: Improvement of Dynamic Scheduling in homogeneous cluster.

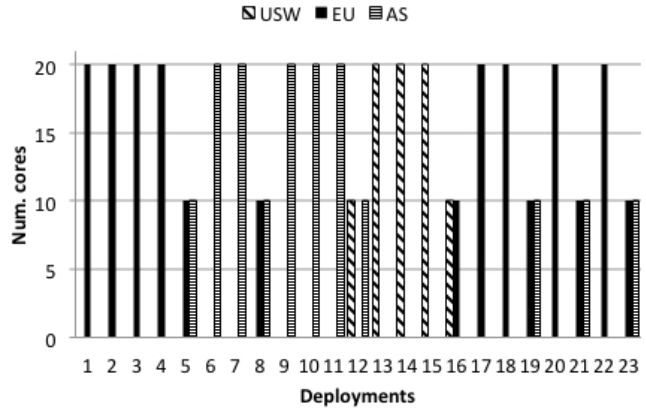
	Scheduling scenario					
	Static	**10 %**	**25 %**	**50 %**	**100 %**	**Oracle**
TIC	14,25 €	14,02 €	13,95 €	13,84 €	13,69 €	13,54 €
Saving	—	1,614 %	2,105 %	2,877 %	3,943 %	4,98 %



(a) Reallocating up to 10 % of the infrastructure.

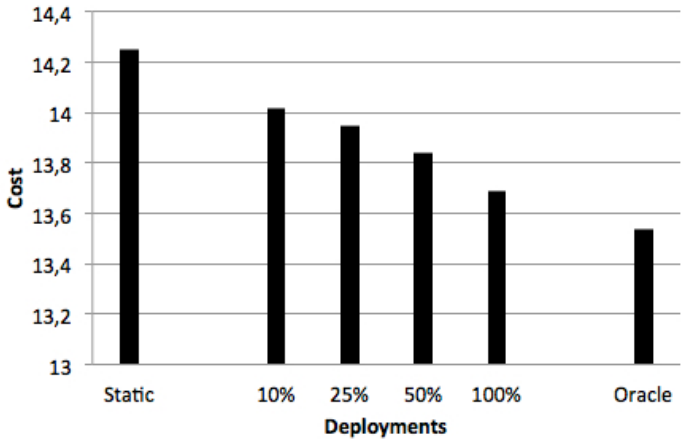


(b) Reallocating up to 25 % of the infrastructure.



(c) Reallocating up to 50 % of the infrastructure.

Figure 4.15: Estimated TIC optimization deployment with different reallocation constraints.



(a) Comparing static, dynamic and oracle deployments.

Figure 4.16: Homogeneous cluster: cost comparison

Within the *heterogeneous cluster* experimental environment, the performance constraint is expressed as the minimum number of cores to reach in each experiment. We define the following requirements:

- Reach a fixed 24-cores performance goal during one day.
- Use all the available instance types if needed.

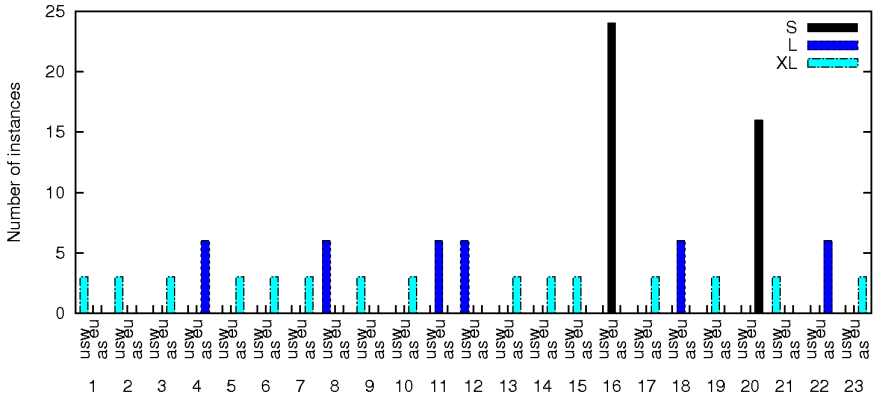
Table 4.6 resumes the constraints applied to our objective function in each case of heterogeneous cluster experiments.

Table 4.6: Constraints used in heterogeneous cluster experiments.

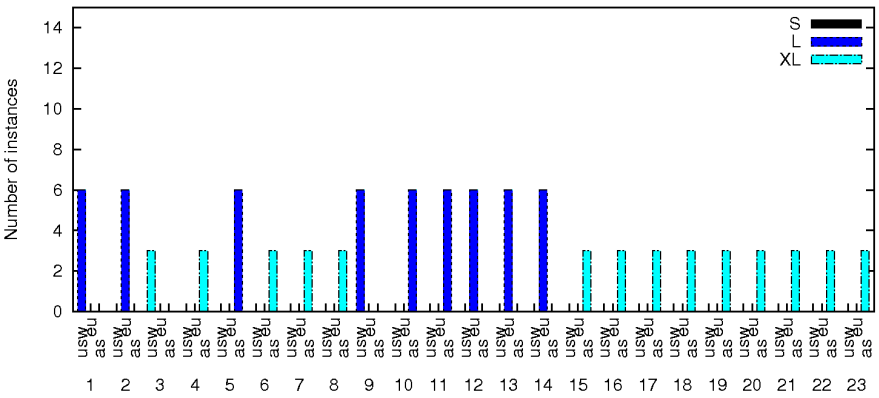
Fig.	**Deploy. Type**	**TIC**	**Constraints**			
			Perf.	**Realloc.**	**I.Type**	**Cloud**
4.17a	Dynamic	Oracle	24 cores	0-100 %	XL (8 cores) L (4 cores) and S (1 core)	USW, EU, and AS
4.17b		Est.		0-100 %		
4.17c		Est.		0-50 %		

Figure 4.17 depicts the comparison of different deployments, such as the oracle TIC and estimated TIC deployments without reallocation restriction applied, and the predicted deployment regarding the possibility of reallocating up to 50 % of the required infrastructure.

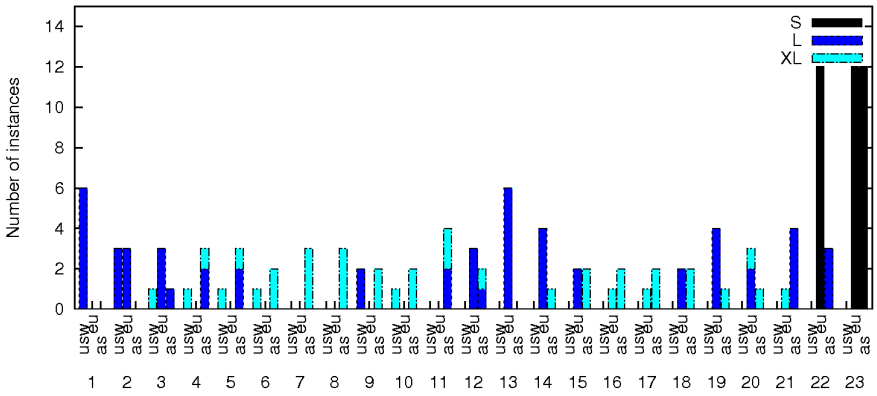
Figure 4.17a shows the Oracle TIC deployment. Here, the broker switches between instance types and cloud providers without any reallocation restriction. Moreover, we observe that the broker uses all the instance types and cloud providers at least once, which means that the best prices can appear in any type of instance within any provider, and therefore the use of a broker makes more sense.



(a) Oracle TIC deployment, all instance types, no reallocation constraint.



(b) Estimated TIC deployment, all instance types, no reallocation constraint..



(c) Estimated TIC deployment, all instance types, 50 % reallocation constraint.

Figure 4.17: Cost optimization deployments regarding the reallocation constraint.

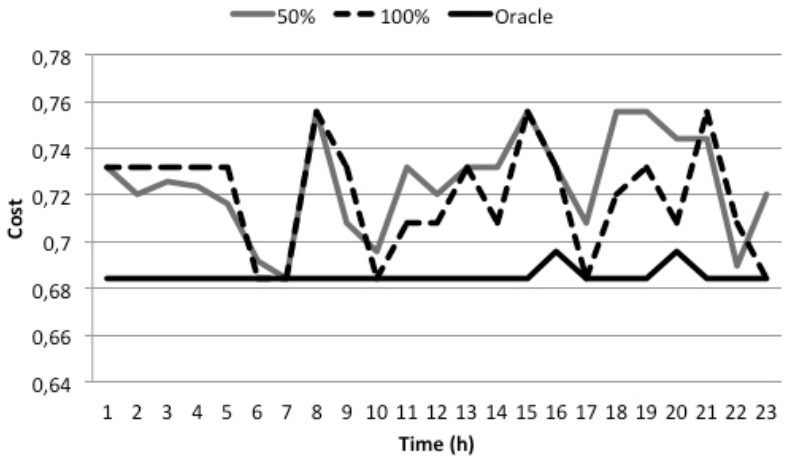


Figure 4.18: Comparing different deployments for heterogeneous cluster.

Figures 4.17b and 4.17c shows the predicted deployments (using the estimated TIC function) when the broker can reallocate the 100 % and 50 % of the infrastructure respectively. Comparing both, we highlight how the broker in 4.17c moves the virtual machines to the predicted best placement (shown in 4.17b), switching from one type of instance to another as required in each moment.

Finally, Figure 4.18 shows the comparison of the hourly cost of these three deployments. Although costs are very similar, results from forecast deployment with up to 100 % of reallocation constraint applied are barely better than the same deployment with up to 50 % of reallocation, but barely worse than the oracle one.

4.2.2. HPC cluster

The adoption of federated clouds for deploying HPC clusters has been studied in several works, such as [MVML11b] or [MMVL11]. A traditional HPC cluster is typically built from many general purpose computers connected together thought networks, and centrally coordinated by some special software. The common term for this is *cluster* because the computers are usually physically very close together. However, in a cloud based HPC cluster, the VMs that replace computers can be distributed in different hosts, different networks, or even different cloud locations. On the other hand, cloud addresses scalability issues that traditional clusters have.

In this experiment we present a performance modelling analysis, in which we consider *FLOPS* as our performance metric. After that, we design and execute experiments for optimizing performance regarding cost constraints.

Moreover, we introduce in these experiments the *high-cpu* medium instance type together with standard instance types (small, large, and extra large). Figure 4.7 depicts these instance types features:

	Standard			**High CPU**
Inst. type	**S**	**L**	**XL**	**Med**
CPU (ECU)	1	4	8	5
RAM (Gb)	1,7	7,5	15	1,7
Storage (Gb)	160	850	1690	350

Table 4.7: Instance types features including Medium instance.

4.2.2.1. Performance modelling

There are special benchmarks for measuring clusters performance depending on the workload. In our case, we consider execution of a large number of loosely-coupled tasks, without heavy network inter-dependency. This type of computing is also associated to high throughput clusters (HTC). The unit of measure of these benchmarks is usually FLOPS (floating point operations per second). For the performance analysis we have chosen the well-known *Linpack benchmark* [DLP03] because it is widely used, and its performance measurements are available for almost all relevant systems. The test consists on executing this benchmark in different instance types, and collect the benchmark output results.

Table 4.8 shows the results applying the benchmark over different Amazon EC2 instance types. In this table, CPU is measured in ECUs (EC2 Compute Units), performance is measured in GFLOPS, and the performance-cpu ratio is expressed as the performance expected per ECU.

	Standard			**High CPU**
Ins. type	**S**	**L**	**XL**	**Med**
Cpu	1	4	8	5
Perf	3.55	11.57	18,32	13.99
Ratio	3,55	2,89	2,29	2,79

Table 4.8: Instance types performance under Linpack benchmark.

As a first result of this table, ratio values show that big instances performs worse that their equivalent in number of smaller ones, although theoretically the performance should be the same or very similar.

4.2.2.2. Performance optimization with cost restriction

The goal of this experiment is to evaluate the performance optimization function for dynamic multi-cloud scenarios in a real use case (a HPC cluster) with different cost restrictions, and also to show how the application-based performance metrics can be useful to optimize the broker deployments.

We consider the following constraints to our objective function:

- Cost: a fixed hourly budget of 1 €,
- Placement: one particular cloud vs. all available ones;
- Instance type: one particular instance type vs. all available ones.

And the following parameters:

- A scheduling period of 24 hours
- The performance metrics of Table 4.8, which are used as broker performance input.

We divide this experiment into two approaches:

a) Experiment 1

In the first approach we use a single type of instance, and we compare the performance achieved with this type of instance in a single cloud, against a multiple cloud deployment. We repeat the experiment for each instance type.

Figures 4.19, 4.20, and 4.21 shows that the multi-cloud solution overperforms the other single-cloud ones, in each instance type case. We also observe that using small instance types gets better performance than others. This is because the small instance type adjusts the price to the threshold better than others.

Figures 4.22, 4.23, and 4.24 shows the performance improvement percentage that the broker gets compared to each single cloud deployments, and also the extra-cost that the broker causes to reach this performance. In these cases, the more performance the infrastructure reach, the more expensive the infrastructure is, always under the defined budget constraint.

b) Experiment 2

In the second approach we use all the available instance types, and we compare again the performance of single cloud deployments (static) with the performance of a multiple cloud deployment (dynamic).

The performance comparison of Figure 4.25 shows two important facts: First of all, the total performance achieved using different instance types always improves the single-instance-type performance, both in the single-cloud

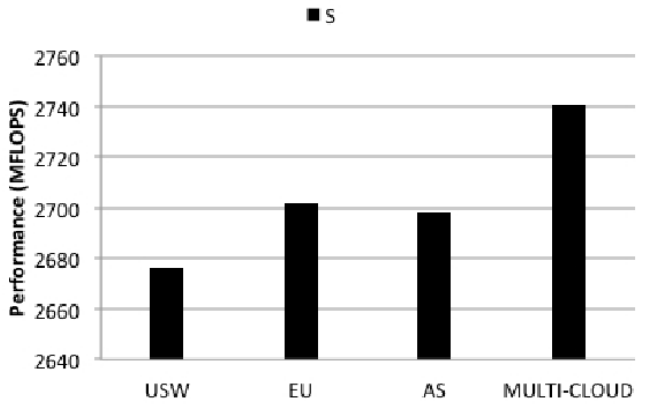


Figure 4.19: Performance: S instance type in a single cloud vs. multiple clouds.

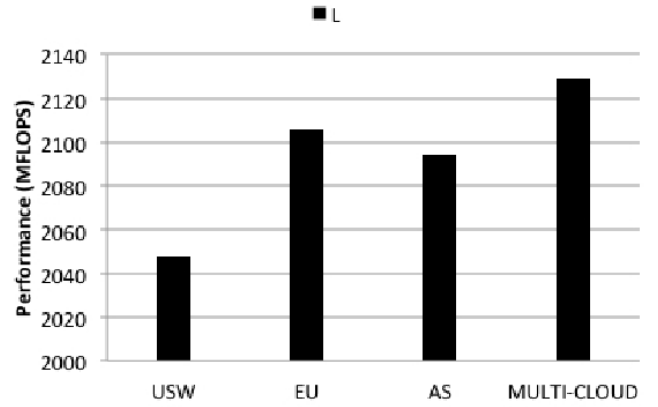


Figure 4.20: Performance: L instance type in a single cloud vs. multiple clouds.

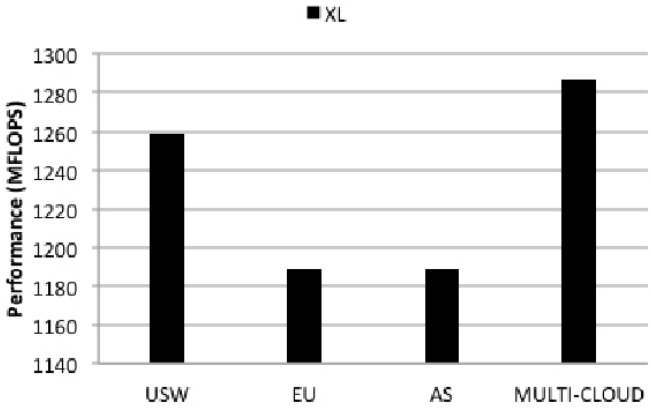


Figure 4.21: Performance: XL instance type in a single cloud vs. multiple clouds.

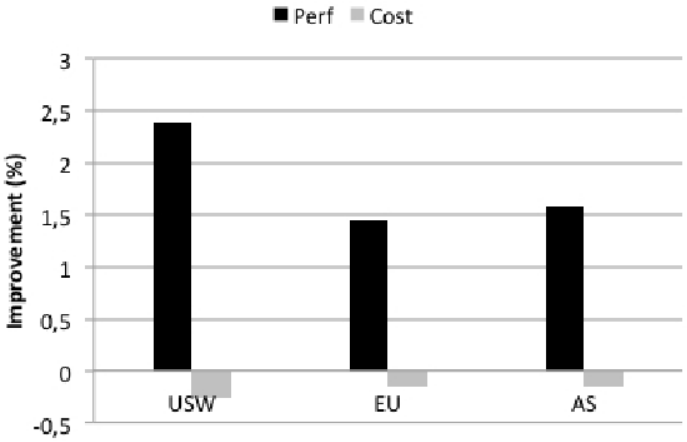


Figure 4.22: Performance and cost comparison. Small IT in single cloud vs multiple clouds

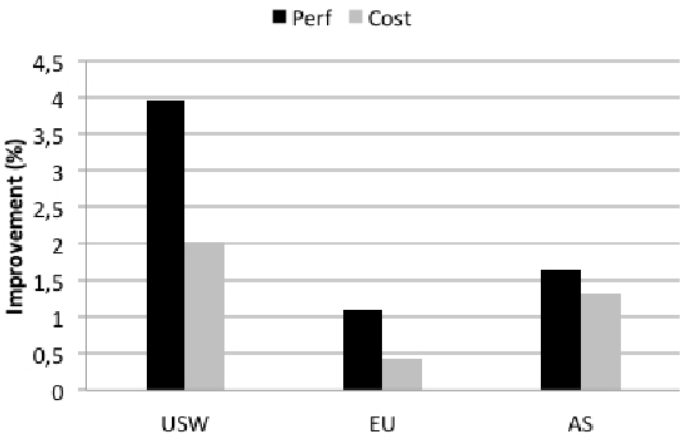


Figure 4.23: Performance and cost comparison. Large IT in single cloud vs multiple clouds.

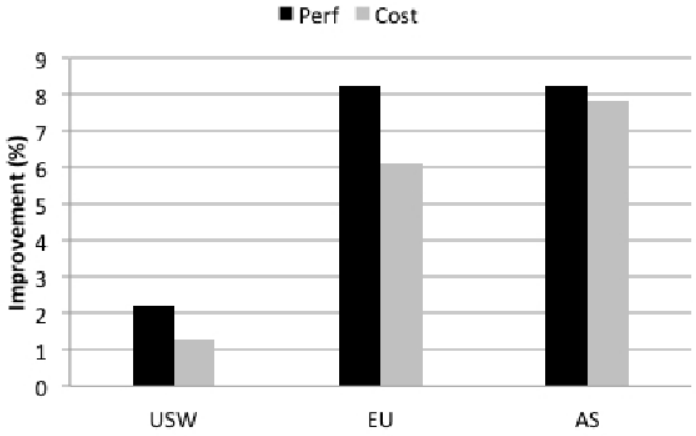


Figure 4.24: Performance and cost comparison. Extra Large IT in single cloud vs multiple clouds.

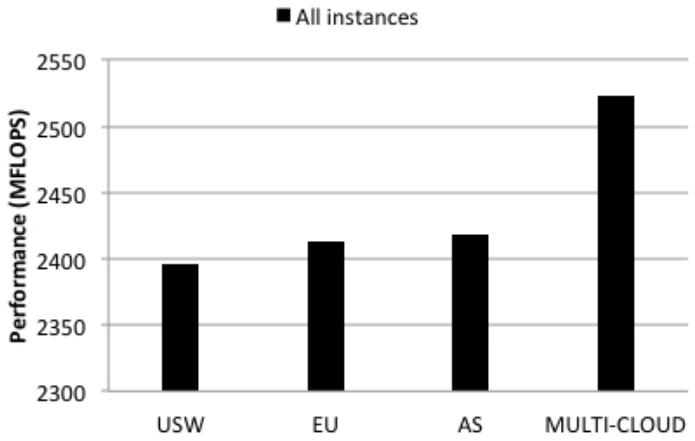


Figure 4.25: Performance comparison: different instance types in single vs. multiple clouds.

cases and the multi-cloud case; and also, in case of using all the available instances, the multi-cloud deployment performs better than single cloud deployments.

Finally, focusing on the cost/performance comparison of Figure 4.26, we observe that in two of these cases the broker optimizes the performance in almost 4 % and 3 % with almost the same cost, and in the other case the performance is improved in more than 2 % but saving some money.

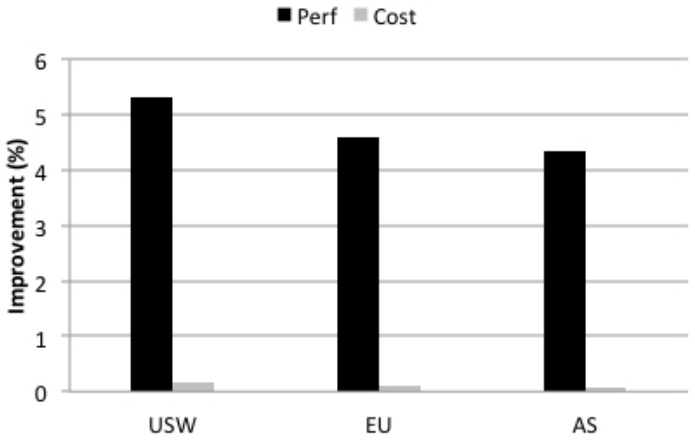


Figure 4.26: Cost and performance improvement: different instance types in single vs. multiple clouds.

4.2.3. Web server

As HPC clusters, Web servers are platforms that researchers from public and private industry are very interested in. In conjunction with Cloud computing and multi-cloud scenarios, Web servers has been analysed in several research works [MVML11a].

Web servers are commonly built using three tiers: a front-end server for static (and potentially cached dynamic) contents; a worker server that process and generate dynamic content; and a back-end database system that manages and provides access to the data.

In this experiment we present a performance modelling analysis, in which we consider *request per second* as our performance metric. After that, we design and execute experiments for optimizing performance regarding cost constraints.

4.2.3.1. Performance modelling

We can measure Web server performance by monitoring different indicators, such as completed request per second, time to wait for a request, or number of request errors. A well-known way of measuring the performance of a Web server is the number of accesses affordable during a period of time. In terms of benchmarking, it means the number of requests per time unit that the server can afford. If the system receives more requests than it can serve, some of them will be enqueued, and therefore a high amount of enqueued requests cause time-out errors.

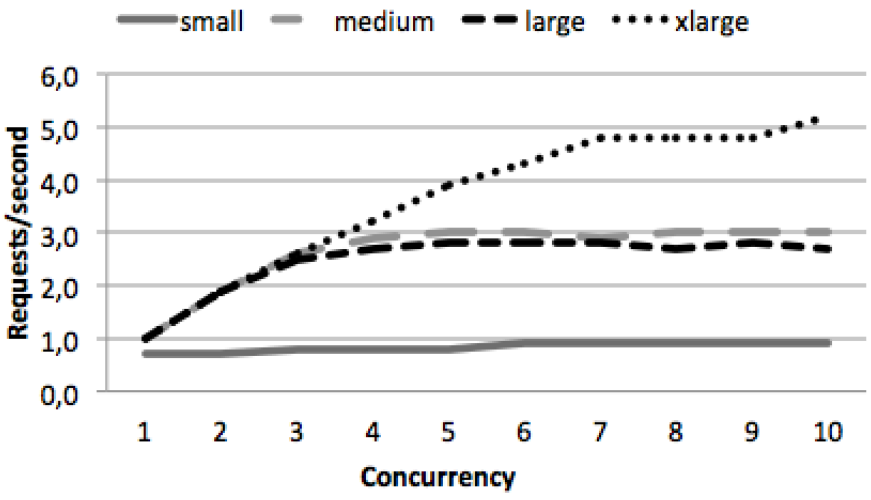


Figure 4.27: Requests attended per second.

Our test consists on the generation of a workload against one Amazon

EC2's instance type that acts as a worker node. And each request consist on solving the *bubble sort* algorithm applied to a fixed-size vector. We use *Nginx* as the Web server installed in the worker node, and *httperf* [5] as the load generator.

Figure 4.27 shows the instance types performance under different concurrent load, measuring requests attended per second.

Figure 4.28 depicts the time taken to attend one request.

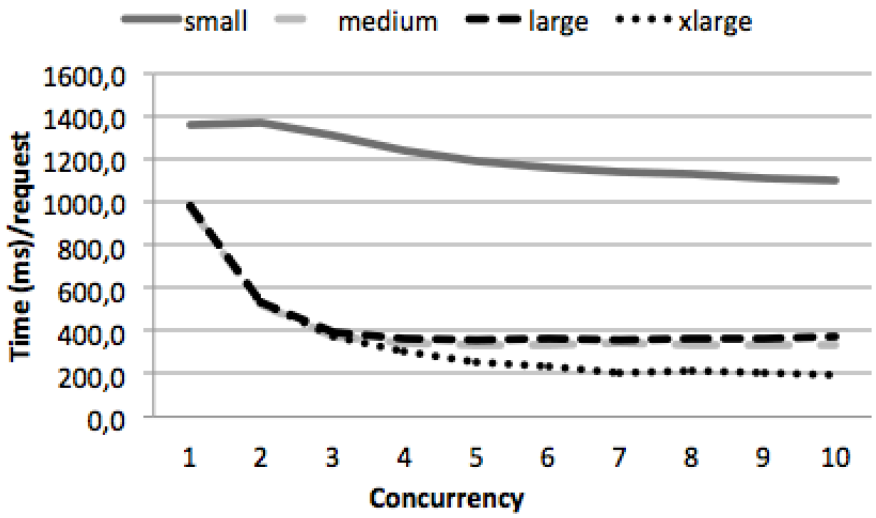


Figure 4.28: Time to attend a single request.

While request concurrency increases, both figures exhibit that every instance type gets overloaded at a certain level: in Figure 4.27 when the requests attended per second do not increase; an in Figure 4.28 when the system can not serve any request in a lower time.

Focusing on Figure 4.27, the small instance maintains almost the same performance during the whole test. It is an evidence that the instance can not solve more requests per second, queueing them when necessary. However, medium and large instances start the test giving nearly the same performance as the small one but they increase its performance when concurrency increases.

Apart from that, both instance types gets overloaded with a similar level of simultaneous requests, showing the same trend than the small instance shows, but with a higher requests rate. Finally xlarge instance performance increases along the whole test having to send more than ten requests simultaneously for getting it overloaded.

Figure 4.29 shows the number of failed requests during the tests, which are those request not attended within the time-out period, regarding different level of concurrency. Medium, large, and xlarge instance types did not show

[5] Httperf home page - `http://www.hpl.hp.com/research/linux/httperf/`

any failed request during the test, but the small instance type did not perform well in cases when concurrency goes up to two requests per second.

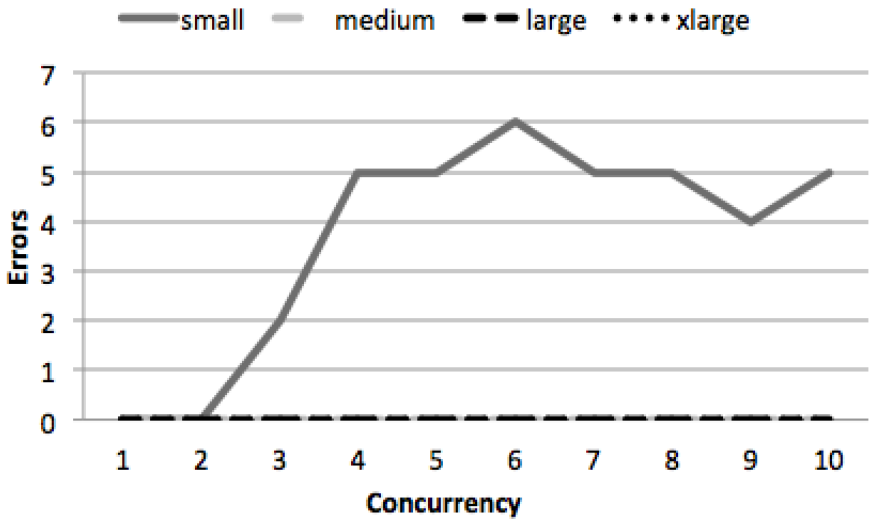


Figure 4.29: Failed requests.

Finally, in next experiments we use the performance results shown in table 4.9, which summarizes the results for the application tested, being *requests per second* the performance metric.

	Standard			**High CPU**
Inst. type	**S**	**L**	**XL**	**Med**
Req/sec	0,7	2,8	5,2	3
Time(ms)/req	1310	360	193	337

Table 4.9: Instance types performance under Nginx Web server.

4.2.3.2. Cost optimization with performance restriction

The goal of this experiment is to evaluate again the cost optimization function for a real use case (Web server), but now using an application level performance constraint. In this experiment, we also evaluate a more complex dynamic scenario, based not only on the variations of the prices but also on the variation of the required service performance.

For this experiment we consider the following constraints to our objective function:

- Performance: a minimum hourly performance to reach, different for each experiment.

- Placement: one particular cloud vs. all available ones.
- Instance type: one particular instance type vs. all available ones.
- Reallocation: reallocate certain a part of the infrastructure.

For the experimentation part, we deploy an infrastructure considering two different requirements:

1 - A *fixed performance*, considering all the possibilities that the broker brings:

   a) Static deployment, in a single cloud provider, and using different combinations of on-demand instances..
   b) Static deployment, in a single cloud provider, and using different combinations of spot instances.
   c) Dynamic deployment (24h period), using spot instances.
   d) Dynamic deployment (24h period), introducing reserved instances to check the broker behaviour under this pricing scheme.

2 - A *dynamic performance* depending on the moment of the day. In this approach we expect the broker to adapt the infrastructure to each hour of the period.

Fixed performance requirement

In experiment a) the goal is to optimize TIC (equation 3.1) in a single deployment with a certain performance requirement. In this case, this performance is a minimum of 13 request per second, and we consider a single cloud provider, on-demand prices, and single instance type deployments against the best combination of them.

Figure 4.30 shows the cost results of these deployments, and the best instance type combination ($3L$ and $2S$) to reach 13 request per second. The solid lines show resources cost using on-demand prices, required performance, and achieved performance. In this figure, S and *multiple-instance* deployments are the best options, since in L and XL deployments the cost is higher than the other cases. As a result, with on-demand prices the broker deployment ($3L$ and $2S$) is as good as the best individual one (S).

In experiment b) we calculate the cost of the same deployments as before, but using spot instead of on-demand prices. Figure 4.31 shows the range of costs of these deployments within a single cloud provider.

The minimum cost of each deployment refers to the lowest spot price of an instance type. Or, in case of multiple instance types, the minimum cost comes from the best combination of lowest spot prices of individual instance types.

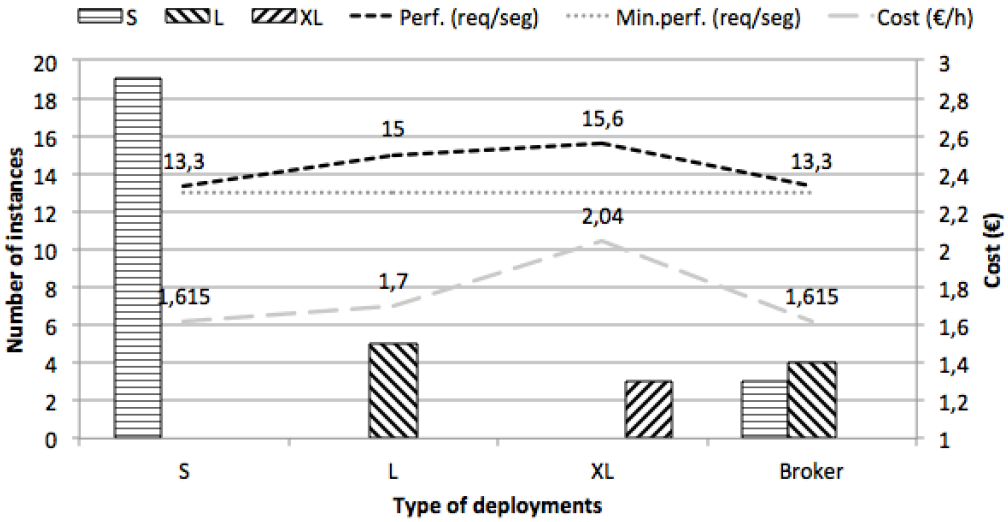


Figure 4.30: Optimize cost regarding a performance constraint in a simple deployment.

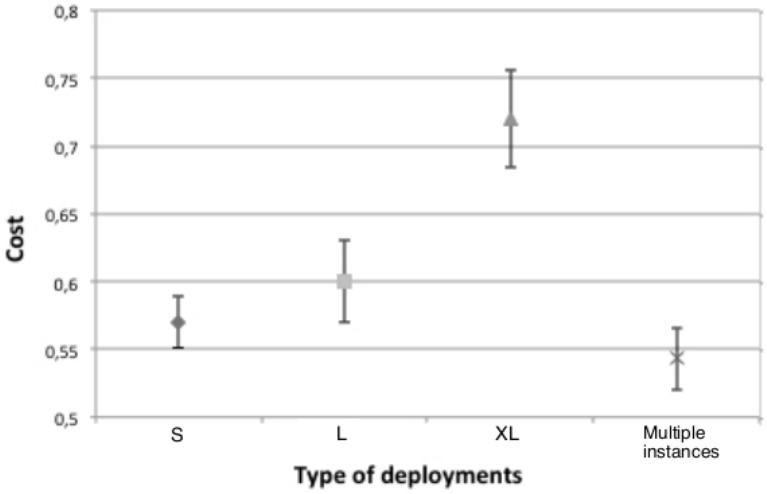


Figure 4.31: Cost using dynamic prices.

Notice that the cost of the deployment using multiple instance types is always cheaper than any other deployment, because it takes advantage of the best combination of instance types in each moment. As a result, considering spot prices, the deployment using multiple instance types ($3L$ and $2S$) is the cheapest one.

Table 4.10 shows the cost improvement of the deployment using multiple instance types, compared to single IT deployments.

	S	**L**	**XL**
Improvement	3,9 - 5,7 %	8,8 - 10,1 %	24 - 25,1 %

Table 4.10: Cost benefit of broker against single IT deployments, using spot prices.

In experiment c), we modify the minimum performance to achieve to 10 request per second. The goal is to observe the different scheduling decisions considering only standard instance types, against introducing the high-cpu medium instance type. Figure 4.32 shows the deployment evolution using standard instance type.

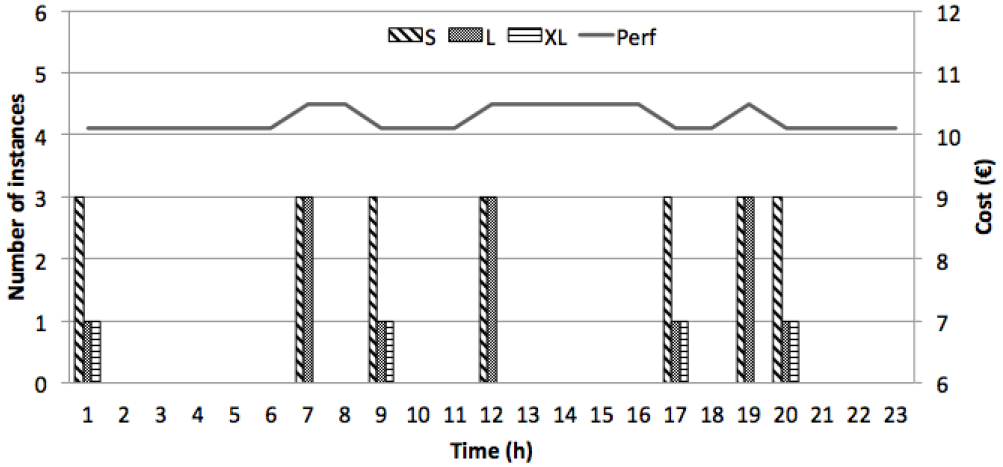


Figure 4.32: Optimal deployment using Standard instances in a 24h period.

Figure 4.33 shows that the broker uses mainly medium instance types, and some small ones to reach the expected performance and optimize the cost. In this case, the broker obtains both better performance results and lower costs than in Figure 4.32.

In experiment d), we consider the same performance requirement as experiment c), but introducing the advance reservation pricing scheme. Figure 4.34 summarizes the cost comparison between deployments of standard and

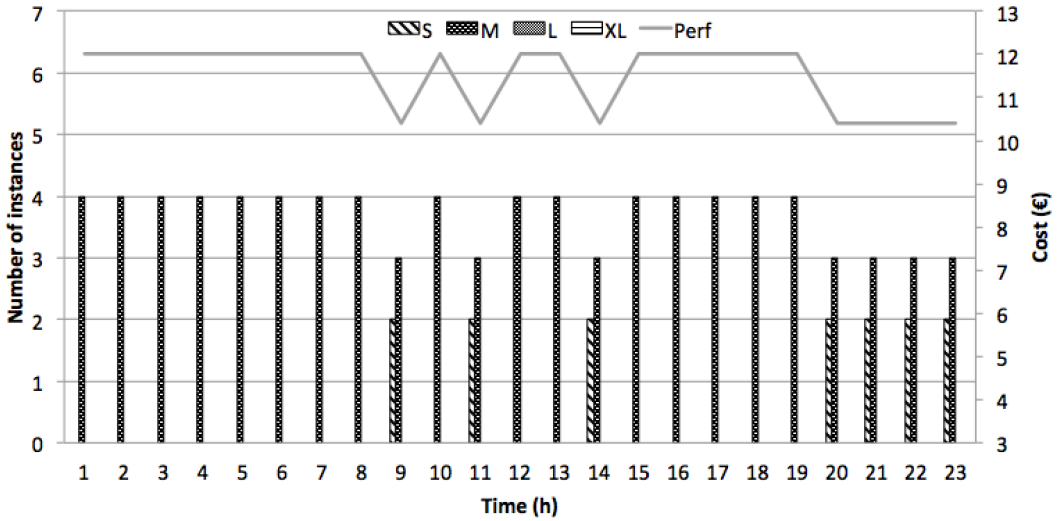


Figure 4.33: Optimal deployment adding Medium instance type.

medium reserved instance types (Std_{rsv} and Med_{rsv} in figure), and deployments of standard and medium on-demand instance types (Std and Med in figure) of Figure 4.31.

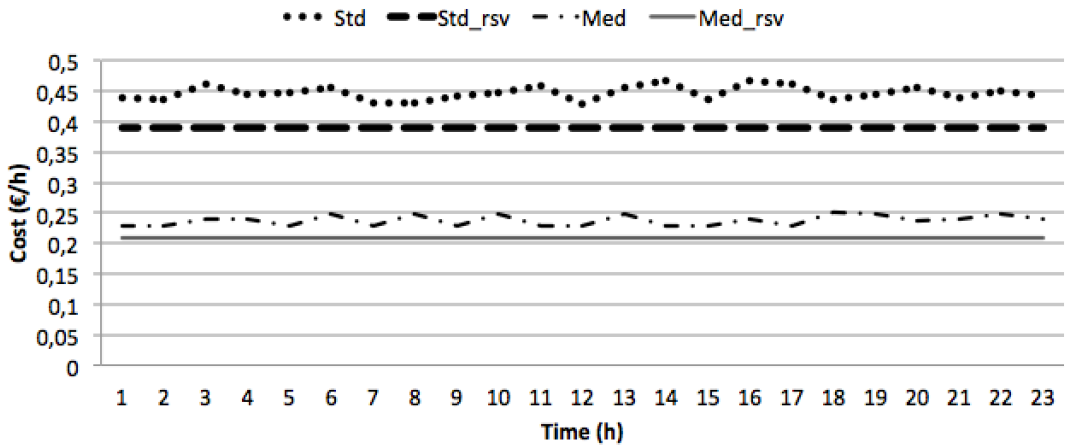


Figure 4.34: Optimal deployment using reserved instances.

As reservation period in Amazon must be for one year at least, we extrapolate to a period of 24 hours. We get the price of reserved instances in one year period: small= 227,50€, large= 910€, xlarge= 1820€, and medium= 455€.

If we reserve the prevailing infrastructure of Figures 4.32 ($3S+1L+1XL$)

and 4.33 ($4M$), in advance, during one year, it would cost 3412,5 € and 1820 € per year respectively, which produces an hourly cost of 0,389 € and 0,208 € respectively. The one-day total cost of these dynamic deployments are 10,287 € and 5,461 € respectively, which means that, in both cases, when the period of use is lower than 11 months, the broker gets better results, but if this period is higher than 11 months the use of reserved instances is justified.

Dynamic performance requirement

In this experiment, the goal is to optimize TIC, but reaching dynamic levels of performance. As an example, we consider the dynamic demand of a company's web server, with high demand in the morning, low demand in the evening, night, and break-times, and medium demand in the afternoon. Figure 4.35 shows the demand profile of this application.

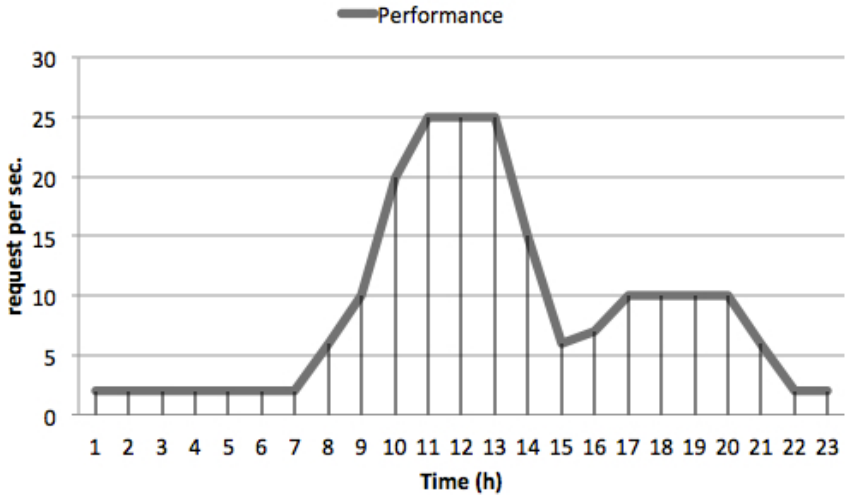


Figure 4.35: Dynamic performance requirements of an ordinary web server.

To control the VMs placement changes, we use the reallocation restriction, together with cloud, and instance type restrictions (equations 3.10, 3.12, and 3.11 respectively) defined in Section 3.2.4.

Regarding Figure 4.35 demand model, next figures depict the optimal deployment when the broker can reallocate the whole infrastructure (Figure 4.36), and when the broker can not reallocate any VM (Figure 4.37).

In the first case, see Figure 4.36, the broker deploys the infrastructure using the cheapest instance types from the cheapest cloud, but in the second case, see Figure 4.37, the broker can not stop any VM to change its placement or the type of the instance, unless the demand makes it necessary. In other

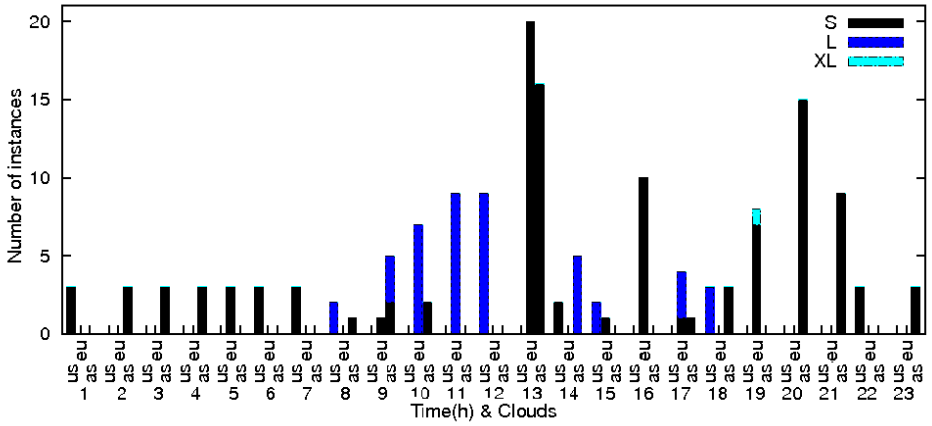


Figure 4.36: Hourly deployments: reallocation constraint + 100 % infrastructure can be reallocated.

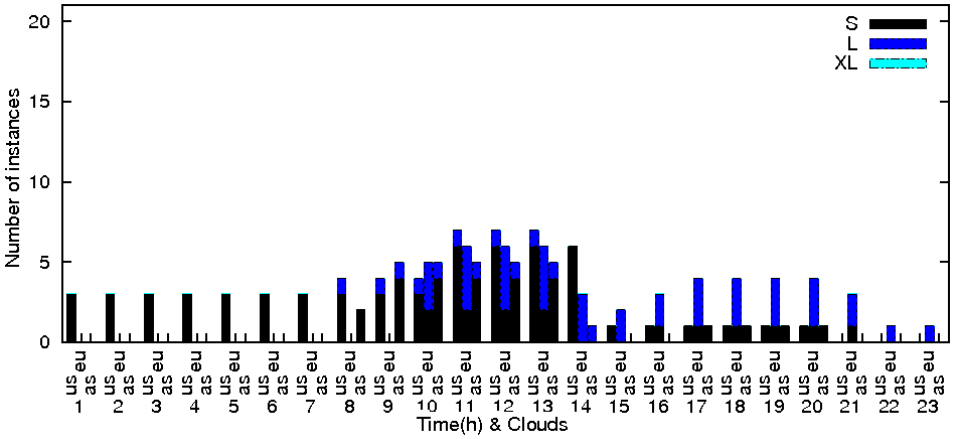


Figure 4.37: Hourly deployments: reallocation constraint + When the broker can not stop any VM unless dynamic demand makes it necessary.

words, when the demand remains similar to the previous hour, the broker can not move any VM. If the demand increases, the broker will deploy more VMs of the best type, but without stopping any existing one. Only when the demand decreases, the broker can stop only the number of VMs which is enough to meet the new demand.

In the first case, the broker provides the cheapest solution of any possible cases, while in the second case the cost is higher but the system does not have performance degradation (as expected), because any VM is stopped unless necessary.

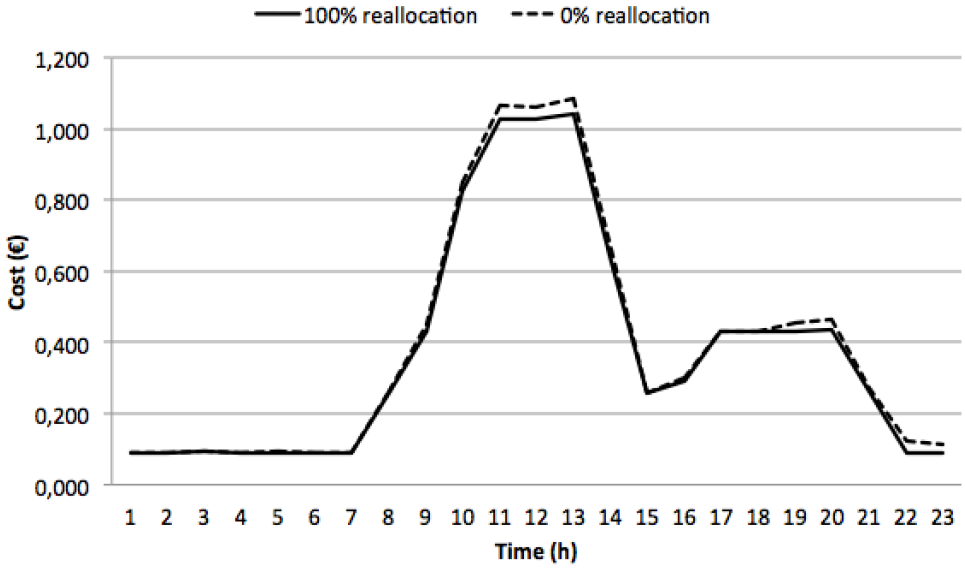


Figure 4.38: Hourly cost of dynamic-demand aware deployments.

Finally, in Figure 4.38 can be observed an hourly cost comparison between Figures 4.36 and 4.37 dynamic-demand deployments. Although both costs seem quite similar, the difference between both cases is close to 4 %.

4.3. Storage-aware brokering

In this section we evaluate the storage proposal explained in Section 3.4, which is about storage strategies considering virtual machine image (VMI) upload, transfer, and deletion policies. As a short summary of Section 3.4, our proposal consists on three strategies:

- Upload the VMI to Everywhere (*E* - in all considered clouds) or On-Demand (*O* - just in selected cloud).
- Delete the VMI Always (*A* - when not used) or Never (*N* - even not

using it).

- Transfer the VMI always from users placement to selected clouds (*Get*), or once from users to the first selected cloud, and then from this one to others (*Copy*).

We analyse the different combinations of Upload and Delete policies (E-A, E-N, O-A, and O-N) considering Get transfer policy, and the same combinations considering Copy transfer policy.

4.3.1. Upload and deletion strategies

The goal of these experiments is to know what is the best combination of VMI storage-deletion policy for different scenarios, and also to know the advantage of using cloud brokering mechanisms considering storage parameters. We define the best combination as the cheapest one, considering both storage and computing costs. Therefore, we apply the cost optimization objective function (equation 3.1).

As an introduction to this first experiment, we want to evaluate the impact of three parameters and its combinations on our algorithms, which are defined as constraints to our objective function:

- The VMI size, considering these possible values: 0.5, 1, 2, and 5 GB.
- The number of VMs required, considering these possible values: 2, 5, and 10 VMs.
- The number of cores required, considering these possible values: 2, 5, 10, and 15 cores.

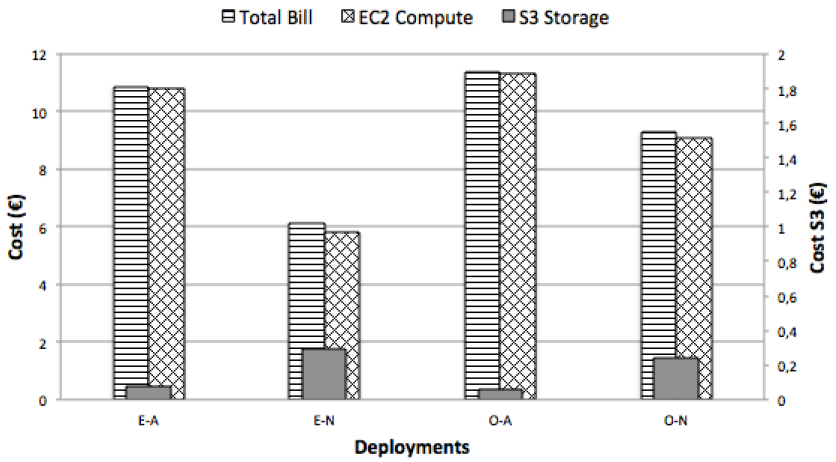
For each triple of parameters, we test each combination of storage and deletion policies, introduced in Section 3.4.

For the first set of experiments, we define the following constraints: 2 VMs with a least 2 cores, and using a 0.5 GB VMI. Obviously, with this combination the instance type constraint become fixed in "small" value, since it is the only instance type that provides 1 core. Neither placement, nor reallocation constraints are applied here.

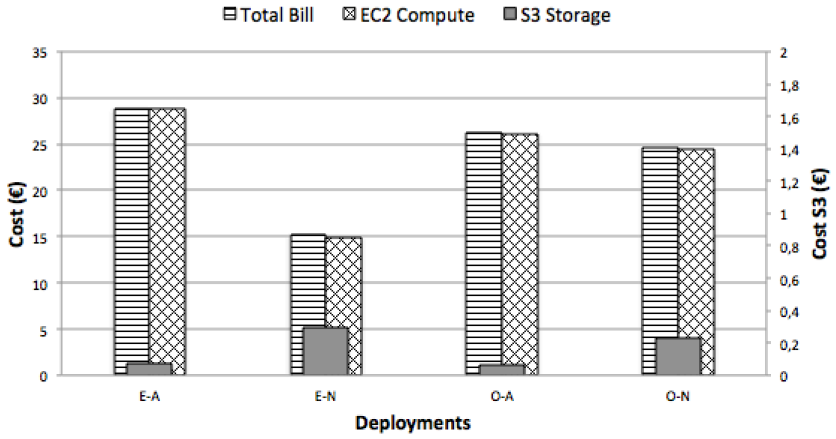
Figures 4.39 and 4.40 show the results of the simulation.

We observe that in this case the best combination of policies is Everywhere-Never (E-N).

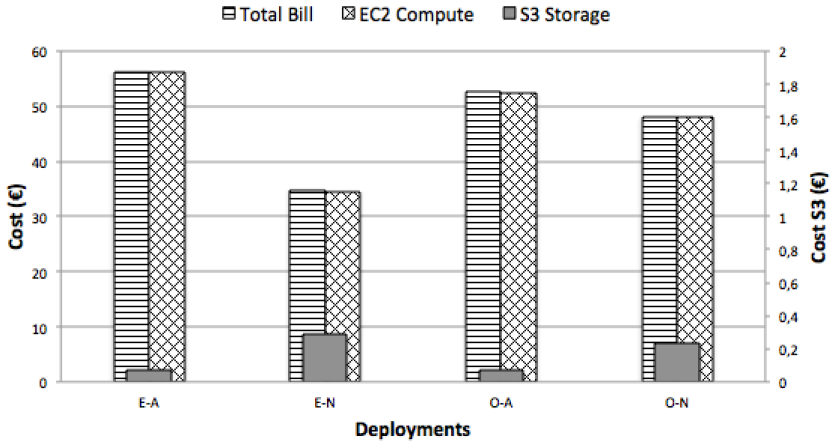
In our previous works we considered cloud proprietary VMIs, which consist just on basic operative systems. The advantage of this type of VMIs is that they are generally offered for free by cloud providers. On the other hand, there are cases in which the VMI must be customized and it can take long time to customize (depending on the packages to install), which at the end it becomes in extra costs.



(a) Cost: 2 VMs 2 cores.

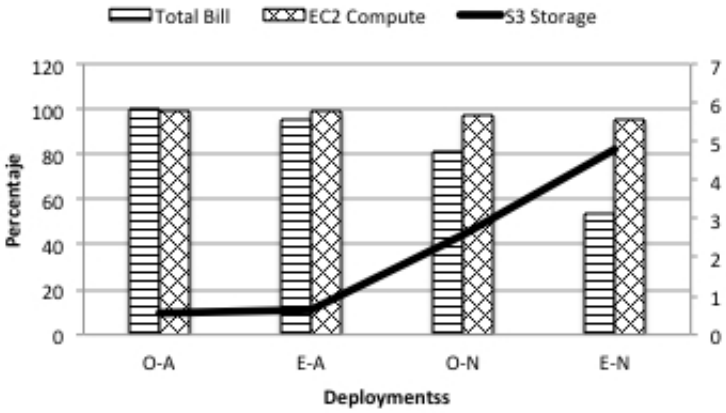


(b) Cost: 5 VMs 5 cores.

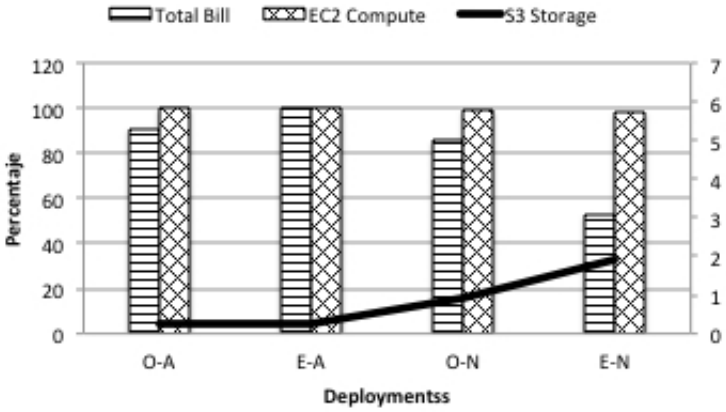


(c) Cost: 10 VMs 10 cores.

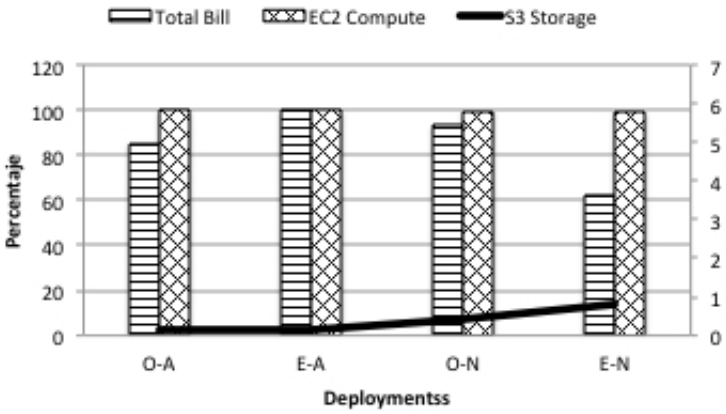
Figure 4.39: Cost results when deploying a Virtual Machine Image (VMI) of 0.5GB.



(a) Percentage: 02 VMs 02 cores



(b) Percentage: 05 VMs 05 cores



(c) Percentage: 10 VMs 10 cores

Figure 4.40: Cost comparison when deploying a Virtual Machine Image (VMI) of 0.5GB.

At the beginning of our experiments we considered as optimal the storage policy based on VMI uploaded on-demand and deleted when not necessary, i.e. O-A policy, in order to not to pay for unused resources. However, we notice that in Figures 4.39a, 4.39b, and 4.39c, with S3 storage, it is clearly better to use the E-N policy.

These results are explained as follows:

- *E-N* is the best solution because once the VMI has been uploaded in every cloud at the beginning of the deployment, the algorithm is aware of the availability of a VMI in each cloud, assumes its cost, and does not take into account the VMI upload cost in the next deployment decisions.

  Therefore, the brokering algorithm focuses only on the best prices of each particular cloud in time t, and it gets full benefits of them by choosing the better one.

  Obviously, storage cost gets clearly incremented, but it is demonstrated that the total cost decreases in every case. Indeed, we can react much faster to changing prices as we do not have the delay of transferring the VMI to a new cloud.

- *O-N* is not as good as the previous one, but its performance gets close to it, as long as the VMI is uploaded in more clouds. Indeed, after some time the VMI tends to be stored in all clouds because of the "never"delete policy, and the situation becomes similar to the aforementioned case (*E-N* policy).

  The decisions with this strategy get conditioned by the fact that the VMI is or is not already available in the cloud with the best price. If not, the algorithm may reallocate the VM in another cloud with a higher compute price, but lower cost considering the VMI upload action.

- For the other combined policies, *E-A* shows better behaviour than *O-A*. This is because once the VMIs have been uploaded, the algorithm can choose the best placement in the first decision without worrying about storage costs.

  Once the first decision has been taken, the VMIs in the unused clouds are deleted, and therefore the next decision will be taken in a similar way in both cases, since both have to upload the VMI again.

Finally, we notice that the storage cost (linked to right axis) remains equal in each case as expected, since the VMI's size is similar in all of them. Moreover, the results show that the deletion policy is more deterministic than the storage policy.

Results from Figures 4.40a, 4.40b, and 4.40c show the percentage of storage cost over the overall cost.

We observe how *N* policies have a higher monetary cost on storage than *A* policies. This is obvious since *A* policies delete the VMI when it is no longer used. Moreover, the more VMs we deploy in the same cloud, the less percentage of storage cost we will have. It can be easily explained as we only need one VMI per cloud whatever the number of VMs we start in each cloud, so the storage percentage decreases as VM number increases.

In Figure 4.41, we present the results of running experiments with different size of VMIs (1GB, 2GB, and 5GB) to see if our previous result with a VMI size of 0.5GB can be applied to bigger VMI size.

It can be observed that the aforementioned best combination of policies (*E-N*) is confirmed as the best one whatever the combinations of VMs, cores selected, or VMI size are. And obviously, the bigger the VMI is, the higher percentage of storage use we obtain.

The following experiments require more cores per VM. Accordingly, small instance type (1 core) is not enough to reach the goal. Hence, the algorithm has to use multiple instance types in these experiments.

For instance, we require 10 cores with 5 VMs in Figure 4.42(a), and 15 cores with 5 VMs in Figure 4.42(b). This fact can lead to use more than one region at once.

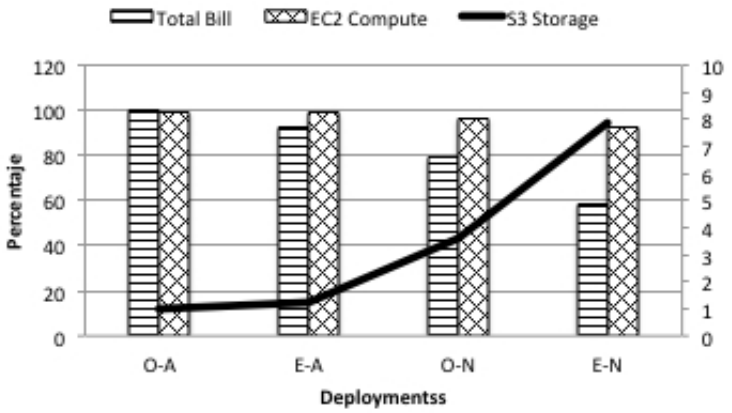
Here, the broker must upload the VMI to different regions, so the total amount of data uploaded can be doubled or tripled in the worst case to achieve 10 cores using 5 VMs (which is 1 XL -4 cores)-, 2 L -2 cores-, and 2 S -1 core-), and deploying them in three different regions. It confirms the intuition we described before: "on-demand" and "always" policies are increasing the start-up time of VMs, and also that, even with larger number of VMs, **E-N is still the best one**.

4.3.2. Transfer strategies

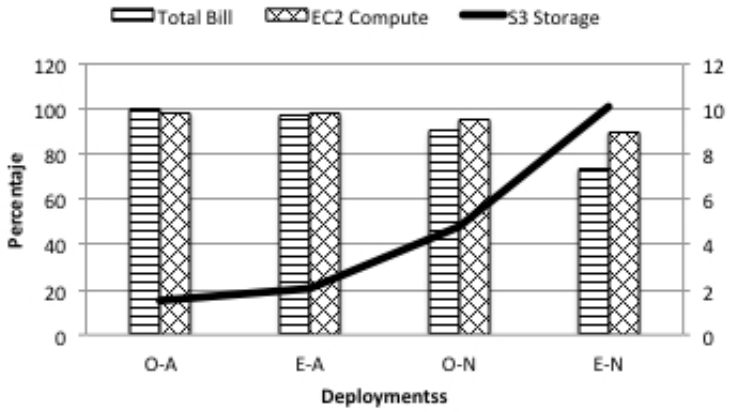
Looking back to Section 4.3.1 experiments, we note that this kind of experiments have some inconveniences. As the VMIs must be uploaded in several clouds, the price could have changed meanwhile the VMs are being started, so it could render unusable the decision taken by the cloud brokering algorithm.

To address this issue, we proposed another way of uploading the VMI, and obviously we implemented it into the simulator. It uses the option offered by Amazon (among others) to *Copy* data from one S3 region to another. Copying files among regions is quicker than upload the VMI from the broker, so once the first upload has been made, it is possible to copy the VMI to the location indicated by the algorithm.

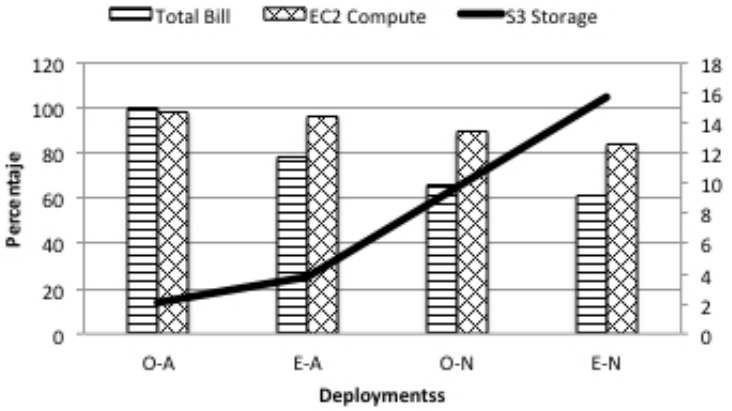
Therefore, the goal of these experiments is to introduce our proposed



(a) Cost percentage: 1GB.

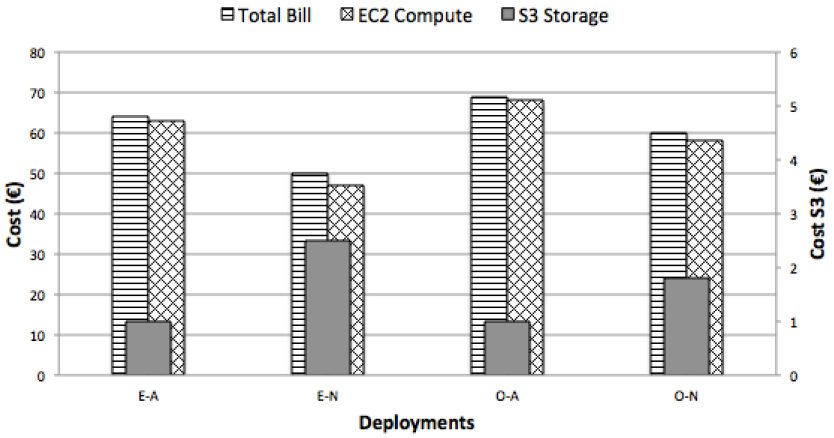


(b) Cost percentage: 2GB.

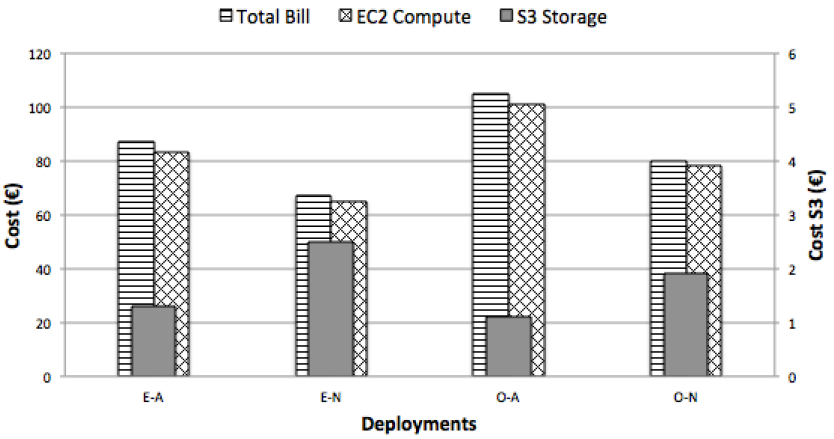


(c) Cost percentage: 5GB

Figure 4.41: Simulation: 2 VMs, 2 cores, different VMI sizes.

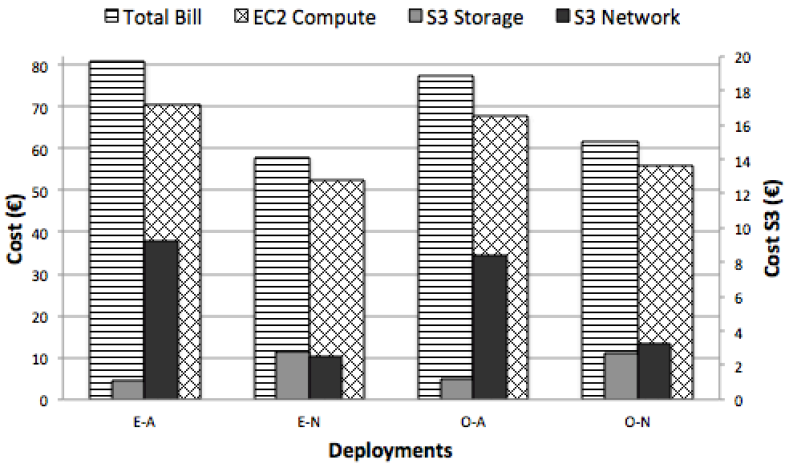


(a) Cost: 5 VMs 10 cores.

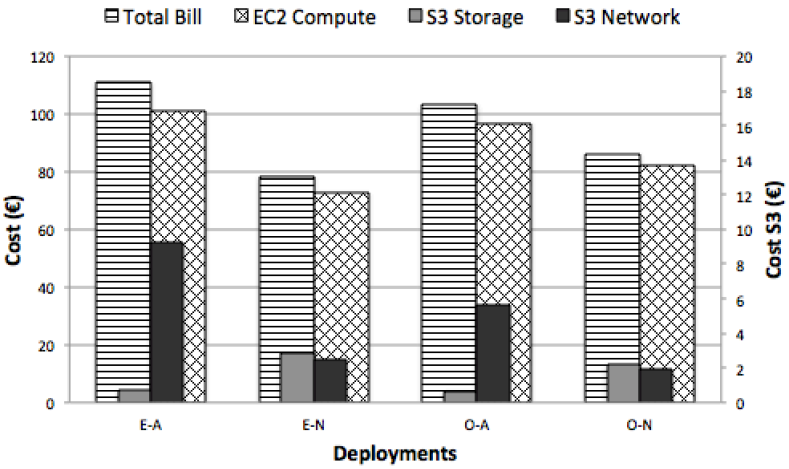


(b) Cost: 5 VMs 15 cores

Figure 4.42: Combination of instance types + get VMI.



(a) Cost: 5 VMs 10 cores.



(b) Cost: 5 VMs 15 cores

Figure 4.43: Combination of instance types + copy VMI.

transfer strategies to study their impact on our algorithms, and to know the cheapest storage policies combination as before. The results are presented in Figures 4.43a and 4.43b.

We observe in both figures that the storage consumption follows the same trends as previously expected. The storage cost is higher in cases where the N strategy is used than cases where A strategy is the one used.

Furthermore, the data transfer cost is higher for the cases that use the A strategy than the ones that use the N strategy as more data is transferred between regions.

4.3.3. Larger experiments

In these experiments our goal is to study the impact of larger VMIs on our algorithms under same conditions (objective function and constraints) as Sections 4.3.1 and 4.3.2.

We still combine the same three parameters as before (increasing its range of values) to test the same combinations of policies. These values are the followings:

- the VMI size (1, 2, 5, 10, 50, 100, 200, 500GB),
- the number of VMs required (1, 2, 5, 10 VMs),
- and the number of cores required (1, 2, 5, 10, 15 cores).

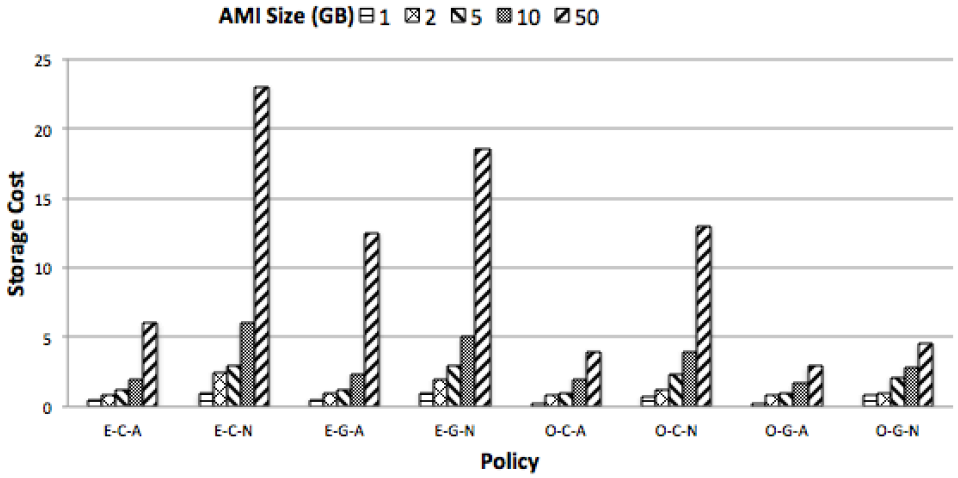
The experiments consist on deploying dynamically (hourly) a certain number of VMs among available cloud regions, changing VM placement when there is a cheaper region, and considering computing and storage costs.

For the first set of experiments, we define the following constraints: 1 VM with at least 1 core. As shown in Figures 4.44 and 4.45, we evaluate all the combinations of storage policies with different VMI sizes.

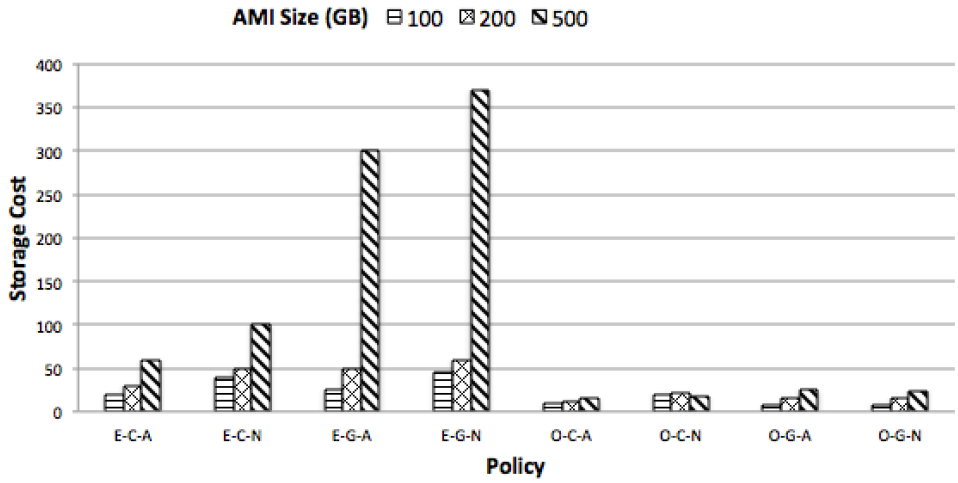
Figures 4.44a and 4.44b show the storage cost, whereas Figures 4.45a and 4.45b present the overall bill.

We observe that the storage cost can decrease whereas the total cost increases, *e.g.* 500GB VMI size for `O_G_A` combination (on-demand,get,always). Accordingly, it is important to study the percentage of the bill due to the storage cost, and not only the storage cost or the total cost. Nonetheless from these figures, we can see that `O_C_A` (on-demand,copy,always) seems to be the most promising combination of policies. Also `O_G_N` (on-demand,get,never) combination is very promising, except for VMI size greater than 100GB.

Figure 4.46 allows a better understanding of the impact of the different combination of policies. It displays the evolution of the percentage of storage cost in the total bill for different combinations of policies with different VMI sizes using the previous constraints, *i.e.* 1 VM with at least 1 core. More than the impact of different policies, the most important thing we learn from this

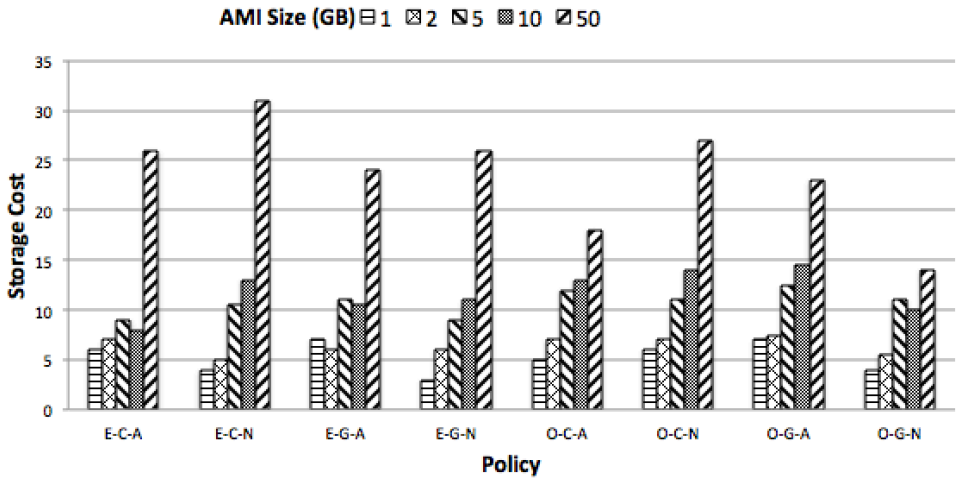


(a) VMI size less than 100 GB.

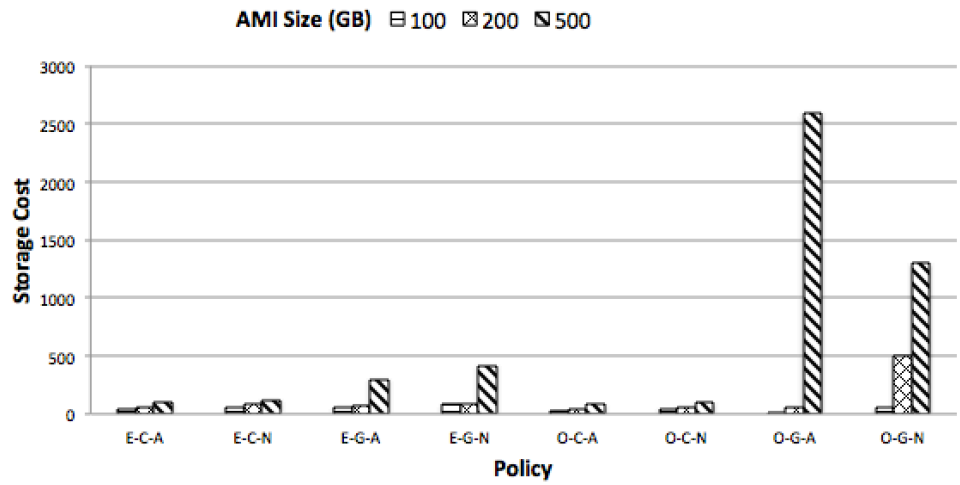


(b) VMI size more than 100 GB.

Figure 4.44: Storage cost for 1 VM and 1 Core with different VMI size.



(a) VMI size less than 100 GB.



(b) VMI size more than 100 GB.

Figure 4.45: Total cost for 1 VM and 1 Core with different VMI size.

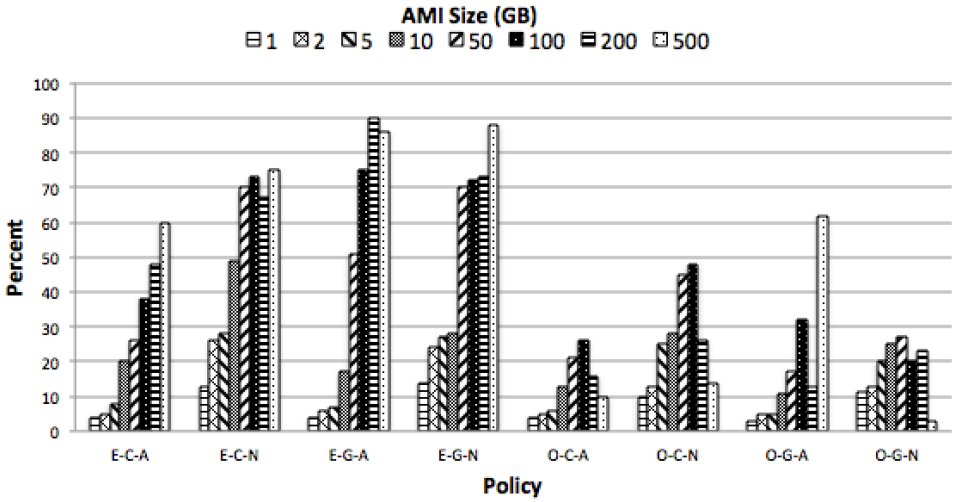


Figure 4.46: Percent of storage cost for 1 VM and 1 core with different VMI size

figure is the importance of storage cost in the total bill. Indeed, most of the previous works neglect storage cost as they state it is very small.

With small VMI size (up to 5GB), it is more or less the case (less than 10 % of the cost is due to storage). Nonetheless, if someone designs an algorithm that save less than 10 % its effects can be eliminated due to storage cost. However, for VMI size greater than 5GB, the storage cost can go up to 90 % of the total bill.

For example, in the case of 200GB VMI, the storage cost can vary from 90 % to 14 % depending of the combinations of transfer, storage, and deletion policies. In these cases, it is mandatory to study the storage cost and how to optimize it, or otherwise other improvements can be render useless.

Therefore, we study in details the impact of different combinations of policies based on the results displayed in Figure 4.46:

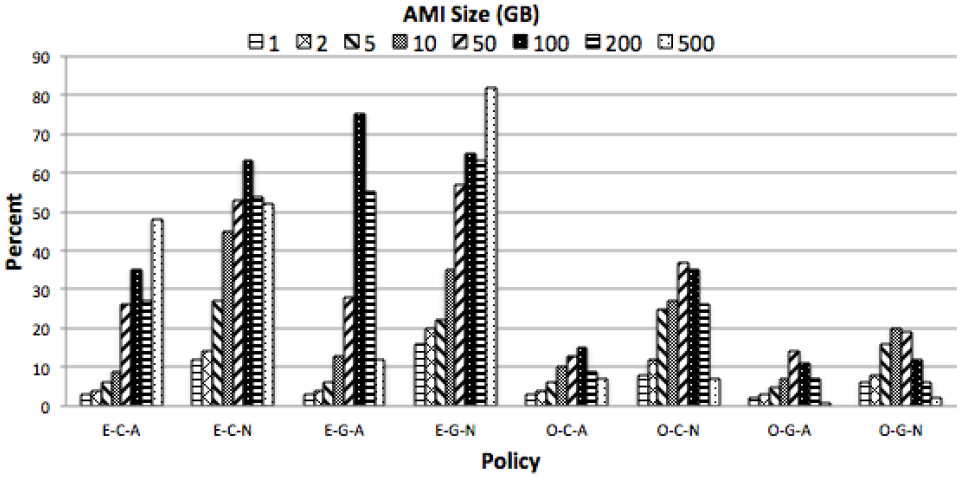
- With the everywhere (E) storage policy, the bill increases with the size of VMI. Moreover, the transfer policy (get or copy) does not have a big impact with everywhere storage policy. Indeed, as the VMI is stored in every region at the beginning, the only difference is due to the network cost, and in these experiments it is very small if not zero (in most cases, uploading to a cloud is free and it is cheap to transfer between different regions/datacenters of the same cloud).

- With the on-demand (O) storage policy, the bill increases with VMI size up to 100GB, and decreases after that in most cases. Moreover, the get transfer policy seems to be a little bit better than the copy one.

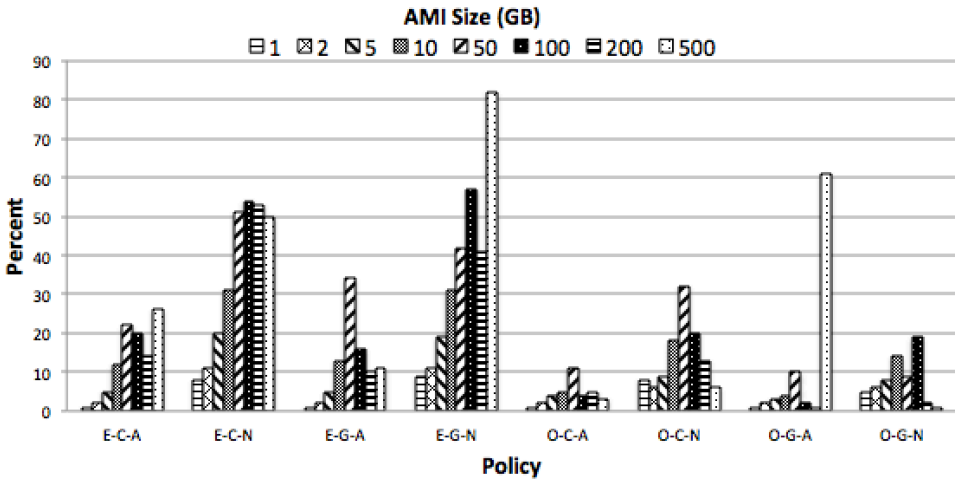
- The `A` deletion policy, compared to `N`, decreases the storage percentage on the bill with all the combinations except for the 500GB VMI.

In every case, on-demand uploading policy is better than any other.

In Figures 4.47, 4.48, 4.49, and 4.50 we study the results of experiments for different number of VMs and cores.



(a) 1 VM and 2 Cores.

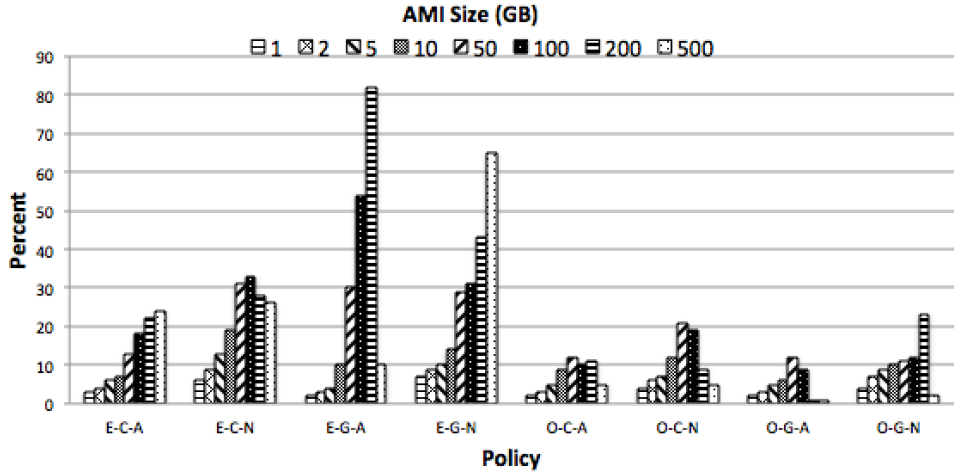


(b) 2 VM and 2 Cores..

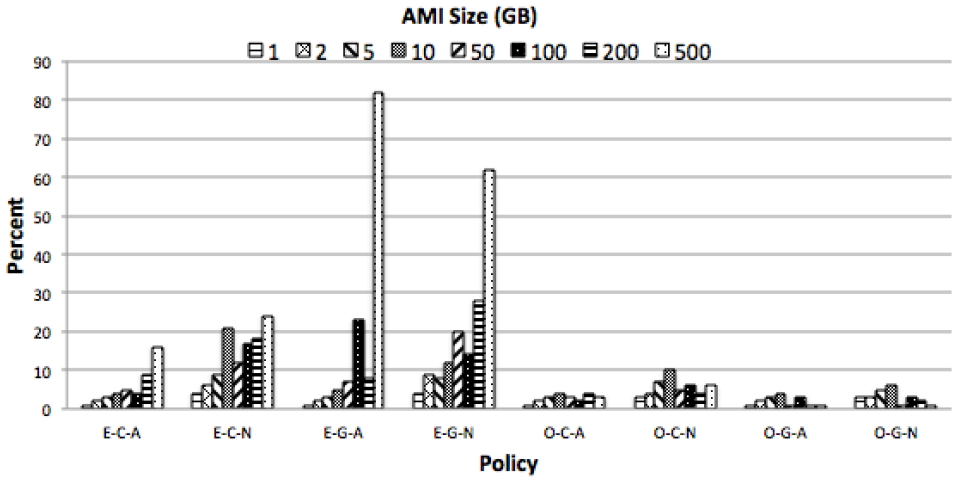
Figure 4.47: Storage cost percentage considering number of cores and VMs, and VMI size -1-.

As we previously stated, the important metric is the percentage due to storage cost on the total bill. Accordingly, we only display this metric for these experiments, and not the storage and total bill contrary to the first experiment.

The main purpose of these experiments is to see if:

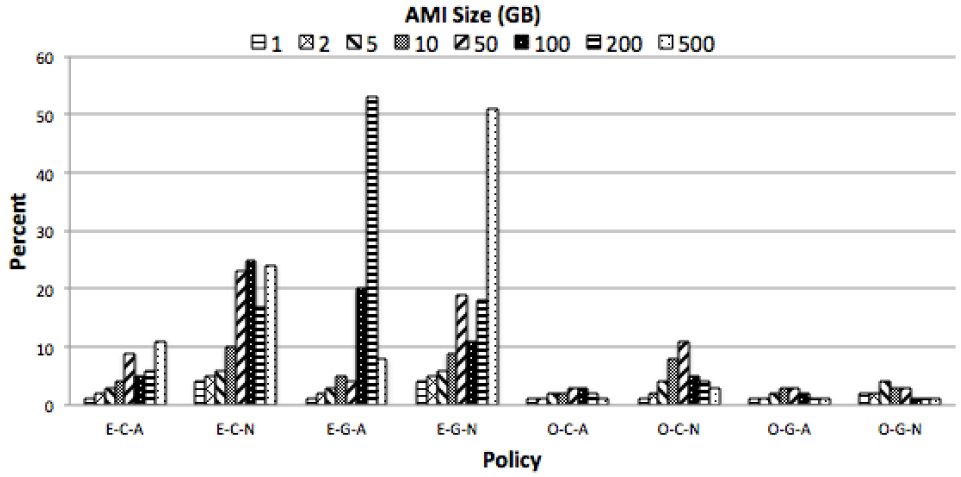


(a) 2 VM and 5 Cores.

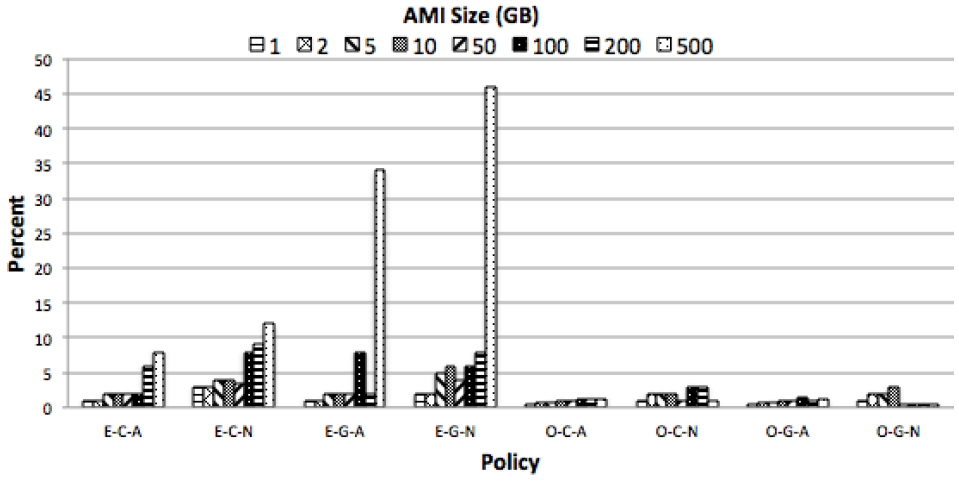


(b) 5 VM and 5 Cores.

Figure 4.48: Storage cost percentage considering number of cores and VMs, and VMI size -2-.

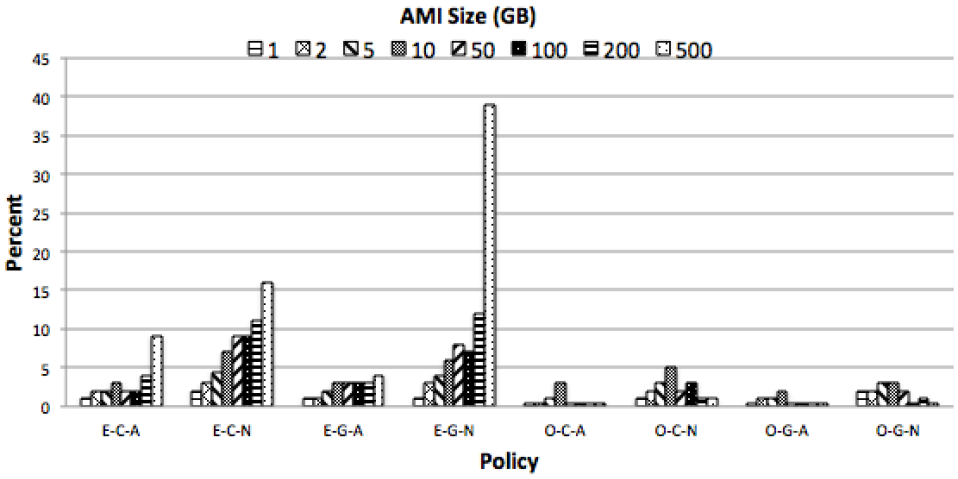


(a) 5 VMs and 10 Cores.

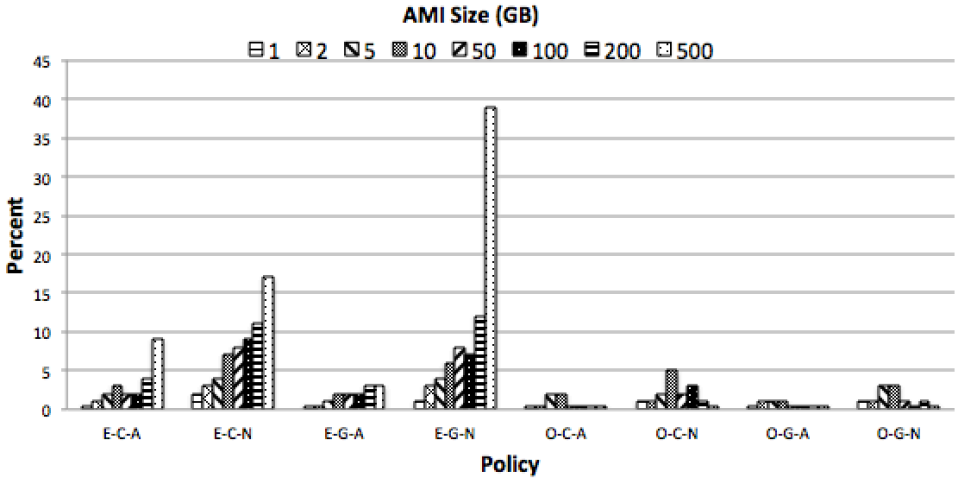


(b) 10 VMs and 10 Cores.

Figure 4.49: Storage cost percentage considering number of cores and VMs, and VMI size -3-.



(a) 5 VMs and 15 Cores.



(b) 10 VMs and 15 Cores.

Figure 4.50: Storage cost percentage considering number of cores and VMs, and VMI size -4-.

- 1) the impact of storage cost on the total bill remains even with more VMs and cores.
- 2) the different combinations of policies are sensitive to the number of VMs and/or cores.

From these figures, it is clear that with more VMs, the percentage of the bill due to storage decreases. It can be explained by the fact that in our experimentation, all the VMs share the same VMI.

Accordingly, the cost of running VMs increases with the number of VMs, whereas the cost of storage does not move. The same observation is true for the number of cores. Indeed, with more cores, the algorithm selects more expensive instance types whereas the storage cost remains the same. Consequently, we state that the number of VMs and cores has a strong impact on the percentage of the storage cost on the total bill.

If an application uses a lot of VMs and cores with the same small VMI, the storage cost can be neglected. Nonetheless, for all the other combinations of parameters (VMs, cores, VMI size), the storage cost has an impact of at least few percentage that can render some optimization useless.

Finally, the observation between the different combinations of policies remains true whatever the number of VMs and cores requested.

Chapter 5

Principal contributions and future work

La conclusión es que sabemos muy poco y sin embargo es asombroso lo mucho que conocemos. Y más asombroso todavía que un conocimiento tan pequeño pueda dar tanto poder.

Bertrand Russel

In this Chapter, we conclude our work by summarizing the results obtained in the experiments. First, we remark the experimental results extracted from Chapter 4, giving a short explanation of them. After that, we remark the non-experimental contributions of this thesis: a brokering architecture for multi-cloud environments, different brokering algorithms for deploying virtual infrastructure while optimizing certain infrastructure parameters, and the contribution to the SimGrid Cloud Broker simulator.

In next sections we mention some future lines of work that we consider appropriate. To conclude this chapter, we give the list of publications that we achieved during this thesis, which validate the results of this thesis within the scientific community.

5.1. The final remarks

As final remarks, in this section we summarize the experimental results, in the same order as we explained our experiments. These are the main conclusions:

- In our first static scheduling experiments, we demonstrate that users can find an optimal deployment if they know in advance the exact amount of hardware that they need. We show how the broker helps users to know in which cases (under determined combinations of CPU, RAM, and HDD) certain cloud providers are better that others. In results of Section 4.1.1, although Amazon EC2 is the provider more popular, we demonstrate that other providers like ElasticHosts or OpSource are cheaper in certain situations (saving 20 % and 52 % if choosing OpSouce and ElasticHost respectively, instead of always Amazon).

  In the first dynamic scheduling experiments, we introduce the spot pricing scheme and the oracle prices. From the experiments with different clouds and dynamic spot prices, we conclude that moving resources between clouds, instead of maintaining resources in the same cloud, can save up to 5 % per day. In this case we consider a fixed instance type, being very low the price difference of this instance type between clouds. In environment with higher price variability, the economical benefit can be higher.

  In the experiments considering price forecasting methods, we introduce the reallocation constraint, designed to avoid infrastructure performance degradation when reallocating VMs. We also evaluate the propose price forecasting method and its integration with the broker. Hence, we demonstrate that, using the broker to move any percentage of the set of resources from one placement to another, always results in better choice than hold it in a static placement.

- In real **use cases** experimentation, we consider cases such as generic clusters, HPC clusters, and Web server applications. Here we introduce the performance constraint, using number of cores -a standard performance metric- in the first real case, and MFLOPS and request per second, which are both specific application performance metrics, for the other two cases respectively. For the specific application metrics we did a performance measurement study, in order to use these measures as an input for the broker to optimize cost and performance.

  Results from HPC cluster experiments show that multiple instance type deployments out-perform single instance type ones, and multiple-cloud deployments out-perform single-cloud ones, apart from the fact that using the broker facilitates users the development of multi-cloud and multi-instance infrastructure. Only when a single instance type fits exactly to user needs in terms of performance, this deployment can reach as good results as a multi-instance deployment.

  Results from Web server experiments show how the broker performs with static and dynamic performance requirements, in which depending on the moment of the day, the performance requirement can fluctuate.

In static experiments, we show how the broker select the best combination of instance types, and how it reallocates the current infrastructure to reach the performance goal. Hence, we introduce the advance reservation pricing scheme, and we demonstrate that the cloud broker with dynamic prices can improve the benefits of the reservation pricing model. In dynamic experiments, we explain how the broker launch VMs when more performance is needed, and how it does not terminate VMs unless the performance requirement decreases, which guarantees a certain quality of service while optimizing performance parameter.

- Considering **storage** in our experiments, we study through simulation the brokering algorithms under different combinations of virtual machine image (VMI) storage, deletion, and transfer strategies, with different number of VMs and cores.

  In the first experimental results, we highlight the significance of image transfer, deletion and storage policies for multi-clouds environments. Although total cost reduction is quite good, this experiment is mainly focused on select the best strategies and apply them to the brokering algorithm, instead of obtaining cost improvement percentages. Thanks to these experiments, we conclude that keeping images in every cloud during the deployment instead of uploading and deleting them when necessary, results in higher storage costs but lower final bill, which is the goal of the brokering algorithms.

  We also show that, in very few cases, although using an image copy mechanism between clouds is more expensive, the final bill became lower because of the reduced transfer time. Image transfer time concurs in wasting virtual resources time waiting for the images to be uploaded, instead of using these resources for compute.

  In the last experimental results, we highlight the significance of the storage cost on the total bill. We realized that previous works, based on the fact that storage cost can be neglected, were not always enough accurate, so new analysis of them should be done. We show that the impact of storage cost on the total bill (in percentage) decreases with the number of VMs and cores that share the same AMI. Nonetheless, the impact is still not very high (few percentage). However, considering this cost, the result of an optimization can vary from winning to loosing this amount.

  Moreover, in these experiments we consider an AMI that does not change throughout time, and we do not migrate any application states. In real world, such as n-Tier applications or *hadoop* clusters, this is not always the case, for example with data storage components. In these cases, the VMs containing the data storage components must be seen

as VMs with large and unshared -therefore unique- AMI. Consequently, the impact of storage cost would be probable larger. Accordingly, it is mandatory to select the good combinations of policies that can dramatically reduce the bill (in our experiments, from 90 % of the total bill to 14 % of the bill for a AMI size of 200GB).

To conclude, we prove the importance of taking into account storage when designing brokering algorithms and the importance the different storage policies.

5.2. Contributions of this Ph. D. thesis

In this Section we summarize the three main contributions of this Phd Thesis: a cloud brokering architecture, brokering algorithms for performing optimal infrastructure deployments, and the improvements made to the simulator SimGrid Cloud Broker.

- The first contribution is the cloud brokering architecture we presented for deploying virtual infrastructures in static and dynamic multi-cloud environments.

  The aim of this brokering architecture is to be aware of cloud market information, to help users to distribute their services among available clouds making it transparent for them, to help users in the task of managing their virtual infrastructure using a unique interface, and also to provide them a way to optimize some parameters of their service (e.g. cost, performance) with different scheduling strategies.

  This brokering architecture is mainly composed by three components: the Scheduler, the Cloud Manager, and the Information Manager. Among these different components of the broker, we focus this thesis on the scheduling module, which is responsible for optimizing a certain parameters of the service by providing an optimal infrastructure deployment. To optimize the infrastructure deployment, the scheduler is designed to work with brokering algorithms. To help users to define deployment requirements, these brokering algorithms accept constraints, such as placement, instance type or reallocation constraints. Both, algorithms and restrictions, are the second contribution of this thesis.

- The second contribution are the brokering algorithms developed for optimizing several parameters of the required infrastructure. In our proposal, we divide these scheduling algorithms to optimize two main parameters: maximizing the total infrastructure performance (TIP), or minimizing the total cost of the infrastructure (TIC). Both are considered as the possible goals of any deployment, and they can not be satisfied together in the same deployment but isolated.

Users also can restrict the deployment requirements by using the set of constraints proposed in this work. With these restrictions to the algorithms, the broker behaviour is adapted to user needs. For instance, users can indicate their minimum performance expected, their available budget, where to put their resources, the type of resources they want to use, or they can suggest what percentage of their virtual infrastructure is ready to be reconfigured for reaching their performance or cost goal.

Moreover, the algorithms can be improved by adding price forecasting models in case of price optimization. Here, the algorithm takes into account the historical prices of available public cloud providers for the next deployment decision in a dynamic deployment scenario. Using these prices, the scheduler component can analyse and process them, to finally calculate some metrics (such as average or trend) for predicting the best next hour deployment. To summarize, this brokering algorithms and restrictions are the main contribution of this research.

- The third contribution is the improvement done to the simulator SimGrid Cloud Broker (SGCB), which was created for the experimental part of this thesis, thanks to a collaboration with AVALON research group, from Inria. Lyon. However, it was developed and maintained in parallel since its creation.

  We developed our base algorithms and restrictions into SGCB, adapting Choco mathematical library into its code, and we tested the algorithms to make sure that it works as well as our AMPL models. Within this improvement, we developed an extended version of our previous algorithms that takes into account data storage. Although other works exist on cloud brokering algorithms, none of them take into account data storage and transfer. As a result, we proposed two strategies for VMI storage, two strategies for VMI deletion, and also we have introduced two different VMI transfer strategies. To summarize, this simulator is the right tool to continue with this thesis in future research efforts.

5.3. Future lines of work

As the Cloud Computing ecosystem grows and changes in short periods of time, lots of improvements, new features or new environments appear. This means that there are (and there will be) lot of lines to work and research, and at the same time, past research could be obsolete if no one continue with it.

We want to mention the following future lines of work, considering three main aspects: improvements to the current state of this research, research on very close topics that have not been taken into account in this work, and

research on different fields or technologies that can take advantage of our work.

- In the first case, we observed that our algorithms take too long when dealing with a high number of VMs, and the deployment of hundred or thousand of VMs is really interesting in some environments like HPC, or for scalability reasons. Therefore, we plan to work on heuristics that do not explore the full set of solutions looking for the optimal one, but try to approximate it by exploring a reduced subset of them.

- In the second case, we are interested in taking in consideration data transfer costs for brokering mechanisms, better known as network costs. For example, Amazon EC2 bills for many different types of data transfer (i.e, within EC2, data transfers between different data centers, between different regions, or simply from outside Amazon EC2 to inside it and vice versa), and this should be taken into account in case, for instance, of tightly coupled VMs, in which the communication between them is very frequent.

- In the last case, we are interested in testing our storage proposals in big data use cases, such as Hadoop clusters, and applying the brokering algorithms to get benefits in high availability scenarios. In the first one, big data technology works high amounts of data that have to be placed somewhere, so storage is needed here, be moved from some places to another, so network is needed here, and be processed in real time, so infrastructure is needed here. In the second one, high availability scenarios can require to replicate the infrastructure to cope with unexpected failures of currently working infrastructure. In this case, it is assumed that this extra infrastructure will not be used most of the time, and therefore it can be moved from one cloud to another in order to save money, which is one of the objectives of this work.

5.4. Publications

Here is a list with the papers produced during this research:

(1) J.L. Lucas-Simarro, R. Moreno-Vozmediano, R.S. Montero, I.M. Llorente. "Dynamic Placement of Virtual Machines for Cost Optimization in Multi-Cloud Environments". In Proceedings of the 2011 International Conference on High Performance Computing & Simulation (HPCS 2011), Pages 1-7, July 2011. doi: 10.1109/HPCSim.2011.5999800

(2) J.L. Lucas-Simarro, R. Moreno-Vozmediano, R.S. Montero, I.M. Llorente. Scheduling Strategies for Optimal Service Deployment Across

Multiple Clouds. Future Generation Computer Systems, 29(6):1431-1441, August 2012. doi:10.1016/j.future.2012.01.007

(3) J.L. Lucas-Simarro, R. Moreno-Vozmediano, R.S. Montero, and I. M. Llorente (2015), Cost optimization of virtual infrastructures in dynamic multi-cloud scenarios. Concurrency Computat.: Pract. Exper., 27, 2260–2277. doi: 10.1002/cpe.2972.

(4) J.L. Lucas-Simarro, Iñigo San Aniceto, R. Moreno-Vozmediano, R.S. Montero, I.M. Llorente. A Cloud Broker Architecture for Multicloud Environment. In Book: Large Scale Network-Centric Distributed Systems (Chapter 15), 2014. doi:10.1002/9781118640708.ch15

(5) J.L. Lucas-Simarro, R. Moreno-Vozmediano, F. Desprez, J. Rouzaud-Cornabas. Image Transfer and Storage Cost Aware Brokering Strategies for Multiple Clouds. In Proceedings of the 2014 International Conference on Cloud Computing (IEEE CLOUD), Pages 737-744, June 2014. doi: 10.1109/CLOUD.2014.103

Appendix A

Cloud simulators

Simulation has been used extensively in several areas of computer science for decades, e.g., for microprocessor design, or network protocol design. In these areas, several widely used and acknowledged simulation frameworks are available. By comparison, the use of simulation frameworks for distributed applications on distributed computing platforms is not as developed, certainly without any standard simulation tool (although network simulation is a key component of distributed application simulation) [CLQ08].

A key issue in distributed computing is to scientifically assess the quality of several solutions with respect to a particular metric. An obvious approach for obtaining valid experimental results is to conduct experiments on production platforms, or at least on large test-beds, but sometimes this infeasible. In fact, most research results in these areas are obtained via empirical evaluation through experiments.

In real world scenarios, results are often non-reproducible due to resource dynamics (e.g., unpredictable host failures, time- varying or non-deterministic network usage). Even if a stable platform is available, some experiments can only be conducted for the platform configuration at hand, which may not be sufficient to gain all necessary insight in the relative effectiveness of various distributed system or algorithm designs. Moreover, experiments on real-world platforms may be prohibitively time consuming especially if large numbers of experiments are needed to explore many scenarios with reasonable statistical significance. Or extremely expensive, by the way. Given all these difficulties, while researchers always strive to obtain some experimental results in real-world systems, the majority of published results are obtained in simulation [CLQ08].

In the literature there is not a wide range of pure cloud simulators, since they are emerging with the growth of the technology. However, there are few of them that together with extended grid simulators make an wide-enough ecosystem of simulation tools.

Once we introduce the importance of simulation in history, in this work

we consider the following:

- We consider using a cloud simulator in order to perform reproducible experiments, to launch them in an automatic way for avoiding human failures, and to control experiments inputs and outputs. Moreover, we do not need to invest money in experimentation, which is a key point considering public clouds, because small experiments became in high bills. And another key point related to the last one, is that real experiments are very time-consuming (and therefore expensive). Simulating one week of Cloud brokering in less than a minute is essential in this work.

- We choose SimGrid [Web15e], a well known grid simulator and one of the more cited ones in academy, due to the features that it provides, and its big community (members and institutions) and associated tools provided by them.

  Grids and clouds are quite similar, but not exactly the same, so we decided to implement a new simulator on top of it, named SimGrid-CloudBroker (or SGCB, listed in [Web15d]), to be able to experiment with cloud environments taking advantage of the powerful of SimGrid.

In this section, we review different cloud simulators and its features, adding a short summary of why we do not consider using some of them, to end up with an in-depth explanation of SimGrid.

CloudSim

CloudSim is a project from The Cloud Computing and Distributed Systems (CLOUDS) Laboratory of the University of Melbourne, which primary objective is to provide a generalized, and extensible simulation framework that enables seamless modelling, simulation, and experimentation of emerging Cloud computing infrastructures and application services

CloudSim is one of the cited simulators that is based in a previous grid simulator. Some years before CloudSim, the authors created GridSim [Web15b]. The GridSim toolkit is a Java-based discrete-event grid simulation package that provides features for application composition, information services for resource discovery, and interfaces for assigning applications to resources. GridSim also has the ability to model heterogeneous computational resources of varied configurations [BM02].

Going back to the CloudSim project, here are some of its features: The main advantages of using CloudSim for initial performance testing include *time effectiveness*, since it requires very less effort and time to implement Cloud-based application, and to provision test environments, and *flexibility and applicability*, since developers can model and test the performance of their application services in heterogeneous cloud environments with little

programming and deployment effort. CloudSim offers support for modelling and simulation of large-scale Cloud computing environments, including data centers, on a single physical computing node; a self-contained platform for modelling Clouds, service brokers, provisioning, and allocation policies; support for simulation of network connections among the simulated system elements; and facility for simulation of federated cloud environments. Some of the features of CloudSim are the followings: availability of a virtualization engine that aids in the creation and management of multiple, independent, and co-hosted virtualized services on a data center node and flexibility to switch between space-shared and time-shared allocation of processing cores to virtualized services [CRB+11] [BRC09].

However, even the research about what simulator fit our needs, we found that GridSim scaled worse than SimGrid.

The main differences between both simulators are the chosen simulation paradigm, and the threading model. Both SimGrid and GridSim use massively multi-threaded discrete-event based simulation. Under the assumption that network connection is low, there is less need to simulate a network, and it is more efficient to use a discrete-time engine. Whereas SimGrid offers the choice between the use of user space threads and native threads, GridSim only supports native threading, a consequence of the threading model used in current Java Virtual Machines (JVMs).

Looking deeply at both simulators, the differences in simulation paradigm is easily explained by the focus and history of the simulators. SimGrid has a strong focus on accurate network simulation. This accuracy can be achieved best using discrete-event simulation. GridSim started out as a framework for testing resource management policies in grids, and is built on top of the SimJava discrete event engine [DDMVB08].

Finally, we decide not to choose CloudSim for our cloud simulation purposes.

IcanCloud

iCanCloud [Web15c], from The Computer Architecture, Communications and Systems (ARCOS) Research Group, is a simulation platform aimed to model and simulate Cloud computing systems.

The main objective of iCanCloud is to predict the trade-offs between cost and performance of a given set of applications executed in a specific hardware, and then provide to users useful information about such costs [NCVP+11].

The most remarkable features of the iCanCloud simulation platform include the following [NVPC+11]:

- Both existing and non-existing Cloud computing architectures can be modelled and simulated. A flexible cloud hypervisor module provides

an easy method for integrating and testing both new and existent Cloud brokering policies.

- A wide range of configurations for storage systems, which include models for local storage systems, remote storage systems, like NFS, and parallel storage systems, like parallel file systems and RAID systems.
- Customizable VMs can simulate easily both uni-core/multi-core systems using several scheduling policies.
- The memory, storage, and network subsystems can be modeled for simulating a wide range of real systems. Network system can be modelled for simulating a wide range of distributed environments with a high level of detail.
- A user-friendly GUI to ease the generation and customization of large distributed models. This GUI is especially useful for: managing a repository of pre-configured VMs, managing a repository of pre-configured cloud systems, managing a repository of pre-configured experiments, launching experiments from the GUI, and generating graphical reports.
- A POSIX-based API and an adapted MPI library for modelling and simulating applications. Also, several methods for modelling applications can be used in iCanCloud: using traces of real applications; using a state graph; and programming new applications directly in the simulation platform.

ICanCloud is a promising simulation tool, but too young in academy, so the background and community of SimGrid is the main disadvantage we observe to decide not to use it.

GreenCloud

To complete a list of cloud simulators, we also consider GreenCloud. GreenCloud, even though it is not the most suitable simulator for this work, it is one of the simulators that we consider since it is a native cloud simulator, and extending it was a possibility.

GreenCloud [Web15a], from the University of Luxembourg, is a sophisticated packet-level simulator for energy-aware Cloud computing data centers with a focus on cloud communications.

GreenCloud is an extension to the network simulator Ns2 developed for the study of Cloud computing environments. It offers users a detailed fine-grained modelling of the energy consumed by the elements of the data center, such as servers, switches, or links. The simulator is designed to capture details of the energy consumed by data center components as well as packet-level communication patterns in realistic set-ups.

Moreover, GreenCloud offers a thorough investigation of workload distributions. Furthermore, within GreenCloud, a specific focus is devoted on the packet-level simulations of communications in the data center infrastructure, which provide the finest-grain control and is not present in any Cloud computing simulation environment [KBK12].

SimGrid

As we mentioned before, we choose to extend SimGrid simulator for our experiments. SimGrid is a tool-kit providing functions for the simulation of distributed applications in heterogeneous distributed environments. It thereby targets platforms that range from a simple network of workstations to large-scale computational grids. [DMVB09] SimGrid is free software, and its implementation (including a test suite) consists of 10,000 lines of C code. It is distributed under the GPL license, and all developments occurs in open repositories.

More in detail, SimGrid provides a set of core abstractions and functionalities that can be used to easily build simulators for specific application domains and computing environment topologies, which is our case. Some of the key features of SimGrid are [CLQ08]:

- A scalable and extensible simulation engine that implements several validated simulation models, and that makes it possible to simulate arbitrary network topologies, dynamic compute and network resource availabilities, as well as resource failures.

- High-level user interfaces for distributed computing researchers to quickly prototype simulations either in C or in Java;

- APIs for distributed computing developers to develop distributed applications that can seamlessly run in *simulation* or *real-world* mode.

Following with SimGrid internals, it performs event-driven simulation, and the most important component of the simulation process is the *resource modelling*. The current implementation assumes that resources have two performance characteristics: *latency* (time in seconds to access the resource) and *service rate* (number of work units performed per time unit). And an example of resources can be CPUs or network links. For instance, one can create multiple links in between hosts (or group of hosts) to simulate the behaviour of simple routers. This approach is very flexible and makes it possible to use SimGrid for simulating a wide range of computing environments.

Another important component is the *task modelling*. SimGrid provides a C API that allows the user to manipulate two data structures: one for resources (SG-Resource) and one for tasks (SG-Task). An example of task are

data transfers and computations: both are seen as tasks and it is the responsibility of the user to ensure that computations are scheduled on processors and file transfers on network links [Cas01].

SimGrid lets the user describe in different ways a resource or a task. For instance, a processor (resource) is described by a measure of its speed (relative to a reference processor), and a trace of its availability, i.e. the percentage of the CPU that would be allocated to a new process; a network link (resource) is described by a trace of its latency, and a trace of its available bandwidth. A task is described by a name, a cost, and a state. In the case of a data transfer the cost is the data size in bytes, for a computation it is the required processing time (in seconds) on the reference processor.

For our experimentation, as it will explained in Section 4, we define resources such as clouds, regions, or availability zones, and tasks such as image-transfers.

Going back to the explanation, also *traces* play an important role in SimGrid. Simgrid provides mechanisms to model performance characteristics either as constants or from traces. This means that the latency and service rate of each resource can be modelled by a trace. Traces allow the simulation of arbitrary performance fluctuations such as the ones observable for real resources. Moreover, traces from real resources (e.g. CPUs, network links) are available via various monitoring tools. In essence, traces are used to account for potential background load on resources that are time-shared with other applications or users [Cas01].

Finally, more details about SimGrid can be found in some works of SimGrid authors, such as [Cas01], [LQCF06], [Cas01] or [LMC03], among others.

Bibliografía

[AFGJ10] M Armbrust, A Fox, R Griffith, and A Joseph. mOSAIC. Technical report, European Commission: Information Society and Media, May 2010. 2 pages.

[Anu10] Announcing the AWS Asia Pacific (Singapore) region, `http://aws.amazon.com/about-aws/whats-new/2010/04/29/announcing-asia-pacific-singapore-region/`, April 2010.

[Anu14] Announcing the AWS EU Frankfut region, , `http://aws.amazon.com/es/about-aws/whats-new/2014/10/23/announcing-the-aws-eu-frankfurt-region/`, October 2014.

[B+03] Paul Barham et al. Xen and the Art of Virtualization. *SOSP, Symposium on Operating Systems Principles*, pages 164–177, 2003.

[BM02] Rajkumar Buyya and Manzur Murshed. Gridsim: a toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing. *Concurrency and Computation: Practice and Experience*, 14(13-15):1175–1220, 2002.

[BRC09] Rajkumar Buyya, Rajiv Ranjan, and Rodrigo N. Calheiros. Modeling and simulation of scalable cloud computing environments and the cloudsim toolkit: Challenges and opportunities. In *HPCS*, pages 1–11, 2009.

[BRC10] R Buyya, R Ranjan, and R Calheiros. Intercloud: Utility-oriented federation of cloud computing environments for scaling of application services. *Algorithms and Architectures for Parallel Processing*, pages 13–31, march 2010.

[Bro11] Cloud Computing Brokers: A Resource Guide, December 2011.

[BYV08] R. Buyya, Chee Shin Yeo, and S. Venugopal. Market-oriented cloud computing: Vision, hype, and reality for delivering it services as computing utilities. In *High Performance Computing and Communications, 2008. HPCC '08. 10th IEEE International Conference on*, pages 5 –13, September 2008.

[BYV+09] Rajkumar Buyya, Chee Shin Yeo, Srikumar Venugopal, James Broberg, and Ivona Brandic. Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation Computer Systems*, 25(6):599 – 616, 2009.

[Cas01] H. Casanova. Simgrid: a toolkit for the simulation of application scheduling. In *Cluster Computing and the Grid, 2001. Proceedings. First IEEE/ACM International Symposium on*, pages 430–437, 2001.

[CH04] Charles C. and Holt. Forecasting seasonals and trends by exponentially weighted moving averages. *International Journal of Forecasting*, 20(1):5 – 10, 2004.

[CLQ08] H. Casanova, A. Legrand, and M. Quinson. Simgrid: A generic framework for large-scale distributed experiments. In *Computer Modeling and Simulation, 2008. UKSIM 2008. Tenth International Conference on*, pages 126–131, April 2008.

[CRB+11] Rodrigo N. Calheiros, Rajiv Ranjan, Anton Beloglazov, César A. F. De Rose, and Rajkumar Buyya. Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Software: Practice and Experience*, 41(1):23–50, 2011.

[DDMVB08] Wim Depoorter, Nils De Moor, Kurt Vanmechelen, and Jan Broeckhove. Scalability of grid simulators: An evaluation. In Emilio Luque, Tomàs Margalef, and Domingo Benítez, editors, *Euro-Par 2008 – Parallel Processing*, volume 5168 of *Lecture Notes in Computer Science*, pages 544–553. Springer Berlin Heidelberg, 2008.

[DLP03] J. Dongarra, P. Luszczek, and A. Petitet. The linpack benchmark: Past, present, and future. *Concurrency and Computation: Practice and Experience*, 15:2003, 2003.

[DM14] B. Di Martino. Applications portability and services interoperability among multiple clouds. *Cloud Computing, IEEE*, 1(1):74–77, May 2014.

[DMCE15] Beniamino Di Martino, Giuseppina Cretella, and Antonio Esposito. Advances in applications portability and services interoperability among multiple clouds. *Cloud Computing, IEEE*, 2(2):22–28, Mar 2015.

[DMVB09] Silas De Munck, Kurt Vanmechelen, and Jan Broeckhove. Improving the scalability of simgrid using dynamic routing. In Gabrielle Allen, Jarosław Nabrzyski, Edward Seidel, Geert-Dick van Albada, Jack Dongarra, and PeterM.A. Sloot, editors, *Computational Science – ICCS 2009*, volume 5544 of *Lecture Notes in Computer Science*, pages 406–415. Springer Berlin Heidelberg, 2009.

[DRC13] Frédéric Desprez and Jonathan Rouzaud-Cornabas. SimGrid Cloud Broker: Simulating the Amazon AWS Cloud. Technical Report RR-8380, INRIA, October 2013.

[DWC10] Tharam Dillon, Chen Wu, and Elizabeth Chang. Cloud computing: Issues and challenges. *Advanced Information Networking and Applications, International Conference on*, 0:27–33, 2010.

[ea12] Ana Juan Ferrer et. al". Optimis: A holistic approach to cloud service provisioning. *Future Generation Computer Systems*, 28(1):66 – 77, 2012.

[FGK90] R. Fourer, D. M. Gay, and Brian W. Kernighan. A modeling language for mathematical programming. *Management Science*, 36(5):pp. 519–554, 1990.

[FK99] Ian Foster and Carl Kesselman, editors. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1999.

[FLL14] Yuan Feng, Baochun Li, and Bo Li. Price competition in an oligopoly market with multiple iaas cloud providers. *Computers, IEEE Transactions on*, 63(1):59–73, Jan 2014.

[FZRL08] I. Foster, Yong Zhao, I. Raicu, and S. Lu. Cloud computing and grid computing 360-degree compared. In *Grid Computing Environments Workshop, 2008. GCE '08*, pages 1 –10, nov. 2008.

[GGB+15] Mateusz Guzek, Alicja Gniewek, Pascal Bouvry, Jedrzej Musial, and Jacek Blazewicz. Cloud brokering: Current practices and upcoming challenges. *Cloud Computing, IEEE*, 2(2):40–47, Mar 2015.

[GLZ+10] Chunye Gong, Jie Liu, Qiang Zhang, Haitao Chen, and Zhenghu Gong. The characteristics of cloud computing. In *Parallel Processing Workshops (ICPPW), 2010 39th International Conference on*, pages 275–279, 2010.

[HIM02] H. Hacigumus, B. Iyer, and S. Mehrotra. Providing database as a service. In *Data Engineering, 2002. Proceedings. 18th International Conference on*, pages 29–38, 2002.

[JRL08] Narendra Jussien, Guillaume Rochart, and Xavier Lorca. Choco: an Open Source Java Constraint Programming Library. In *CPAIOR'08 Workshop on Open-Source Software for Integer and Contraint Programming (OSSICP'08)*, pages 1–10, Paris, France, France, 2008.

[JTB11] Bahman Javadi, Ruppa K. Thulasiramy, and Rajkumar Buyya. Statistical modeling of spot instance prices in public cloud environments. *Utility and Cloud Computing, IEEE Internatonal Conference on*, 0:219–228, 2011.

[KAACF15] Heba Kurdi, Abeer Al-Anazi, Carlene Campbell, and Auhood Al Faries". A combinatorial optimization algorithm for multiple cloud service composition. *Computers Electrical Engineering*, 42(0):107 – 113, 2015.

[KBK12] Dzmitry Kliazovich, Pascal Bouvry, and SameeUllah Khan. Greencloud: a packet-level simulator of energy-aware cloud computing data centers. *The Journal of Supercomputing*, 62:1263–1283, 2012.

[KF08] Katie Keahey and Tim Freeman. Science Clouds: Early Experiences in Cloud Computing for Scientific Applications. *Cloud Computing and Its Applications*, October 2008.

[KKB+11] T. Kurze, M. Klems, D. Bermbach, A. Lenk, S. Tai, and M Kunze. Cloud federation. *2nd International Conference on Cloud Computing, GRIDs, and Virtualization*, page 32–38, September 2011.

[KKL+] Avi Kivity, Yaniv Kamay, Dor Laor, Uri Lublin, and Anthony Liguori. kvm: the Linux Virtual Machine Monitor. *Linux Symposium*, pages 225–230.

[KKU12] S. Kibe, T. Koyama, and M. Uehara. The evaluations of desktop as a service in an educational cloud. In *Network-Based Information Systems (NBiS), 2012 15th International Conference on*, pages 621–626, Sept 2012.

[KREV+15] D. Kreutz, F.M.V. Ramos, P. Esteves Verissimo, C. Esteve Rothenberg, S. Azodolmolky, and S. Uhlig. Software-defined networking: A comprehensive survey. *Proceedings of the IEEE*, 103(1):14–76, Jan 2015.

[KTMF09] K. Keahey, M. Tsugawa, A. Matsunaga, and J. Fortes. Sky computing. *Internet Computing, IEEE*, 13(5):43 –51, sept.-oct. 2009.

[LMC03] A. Legrand, L. Marchal, and H. Casanova. Scheduling distributed applications: the simgrid simulation framework. In *Cluster Computing and the Grid, 2003. Proceedings. CCGrid 2003. 3rd IEEE/ACM International Symposium on*, pages 138–145, May 2003.

[LMZG15] Xiaoyong Li, Huadong Ma, Feng Zhou, and Xiaolin Gui. Service operator-aware trust scheme for resource matchmaking across multiple clouds. *Parallel and Distributed Systems, IEEE Transactions on*, 26(5):1419–1429, May 2015.

[LMZY15] Xiaoyong Li, Huadong Ma, Feng Zhou, and Wenbin Yao. T-broker: A trust-aware service brokering scheme for multiple cloud collaborative services. *Information Forensics and Security, IEEE Transactions on*, 10(7):1402–1415, July 2015.

[LQCF06] A. Legrand, M. Quinson, H. Casanova, and K. Fujiwara. The simgrid project simulation and deployment of distributed applications. In *High Performance Distributed Computing, 2006 15th IEEE International Symposium on*, pages 385–386, 2006.

[ME11] Thijs Metsch and Andy Edmonds. Open Cloud Computing Interface - Infrastructure. Technical report, OCCI Open Grid Forum, Apri 2011. 15 pages.

[MGR11] Gabriel Mateescu, Wolfgang Gentzsch, and Calvin J. Ribbens. Hybrid computing: "where hpc meets grid and cloud computing. *Future Generation Computer Systems*, 27(5):440 – 453, 2011.

[MMVL11] R.S. Montero, R. Moreno-Vozmediano, and I.M. Llorente. An elasticity model for high throughput computing clusters. *Journal of Parallel and Distributed Computing*, 71(6):750 – 757, 2011.

[MT10] M. Mihailescu and Yong Meng Teo. Dynamic resource pricing on federated clouds. *10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*, 0:513–517, 2010.

[MVML11a] R. Moreno-Vozmediano, R.S. Montero, and I.M. Llorente. Elastic management of web server clusters on distributed virtual infrastructures. *Concurrency and Computation: Practice and Experience*, 23(13):1474–1490, 2011.

[MVML11b] R. Moreno-Vozmediano, R.S. Montero, and I.M. Llorente. Multicloud deployment of computing clusters for loosely coupled mtc applications. *Parallel and Distributed Systems, IEEE Transactions on*, 22(6):924 –930, june 2011.

[MVML12] R. Moreno-Vozmediano, R.S. Montero, and I.M. Llorente. Iaas cloud architecture: From virtualized datacenters to federated cloud infrastructures. *Computer*, 45(12):65 –72, dec. 2012.

[NCVP+11] A. Nunez, G.G. Castane, J.L. Vazquez-Poletti, A.C. Caminero, J. Carretero, and I.M. Llorente. Design of a flexible and scalable hypervisor module for simulating cloud computing environments. In *Performance Evaluation of Computer Telecommunication Systems (SPECTS), 2011 International Symposium on*, pages 265 –270, june 2011.

[NEPM11] Ralf Nyrén, Andy Edmonds, Alexander Papaspyrou, and Thijs Metsch. Open Cloud Computing Interface - Core. Technical report, OCCI Open Grid Forum, Apri 2011. 17 pages.

[NID15] S. Nesmachnow, S. Iturriaga, and B. Dorronsoro. Efficient heuristics for profit optimization of virtual cloud brokers. *Computational Intelligence Magazine, IEEE*, 10(1):33–43, Feb 2015.

[NMN+14] B.A.A. Nunes, M. Mendonca, Xuan-Nam Nguyen, K. Obraczka, and T. Turletti. A survey of software-defined networking: Past, present, and future of programmable networks. *Communications Surveys Tutorials, IEEE*, 16(3):1617–1634, Third 2014.

[NVPC+11] A. Nuñez, J.L. Vazquez-Poletti, A.C. Caminero, J. Carretero, and I.M. Llorente. Design of a new cloud computing simulation platform. In Beniamino Murgante, Osvaldo Gervasi, Andres Iglesias, David Taniar, and BernadyO. Apduhan, editors, *Computational Science and Its Applications - ICCSA 2011*, volume 6784 of *Lecture Notes in Computer Science*, pages 582–593. Springer Berlin Heidelberg, 2011.

[OIY+10] S. Ostermann, A. Iosup, N. Yigitbasi, R. Prodan, T. Fahringer, and D. Epema. A performance analysis of ec2 cloud

computing services for scientific computing. In *Cloud Computing*, volume 34 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, pages 115–131. 2010.

[Pet11] Dana Petcu. Portability and interoperability between clouds: Challenges and case study. In Witold Abramowicz, IgnacioM. Llorente, Mike Surridge, Andrea Zisman, and Julien Vayssiere, editors, *Towards a Service-Based Internet*, volume 6994 of *Lecture Notes in Computer Science*, pages 62–74. Springer Berlin Heidelberg, 2011.

[Pfi98] Gregory F. Pfister. *In Search of Clusters (2Nd Ed.)*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1998.

[PP09] Amit Sheth Pankesh Patel, Ajith Ranabahu. Service level agreement in cloud computing. Technical report, The Ohio Center of Excellence in Knowledge-Enabled Computing (Kno.e.sis), 2009.

[PR14] A.S. Prasad and S. Rao. A mechanism design approach to resource procurement in cloud computing. *Computers, IEEE Transactions on*, 63(1):17–30, Jan 2014.

[RBE+11] B. Rochwerger, D. Breitgand, A. Epstein, D. Hadas, I. Loy, K. Nagin, J. Tordsson, C. Ragusa, M. Villari, S. Clayman, E. Levy, A. Maraschini, P. Massonet, H. Muñoz, and G. Tofetti. Reservoir - when one cloud is not enough. *Computer*, 44(3):44 –51, march 2011.

[SMLF09] B. Sotomayor, R.S. Montero, I.M. Llorente, and I. Foster. Virtual infrastructure management in private and hybrid clouds. *Internet Computing, IEEE*, 13(5):14 –22, sept.-oct. 2009.

[SSMMLF08] Borja Sotomayor, Rubén Santiago Montero, Ignacio Martín Llorente, and Ian Foster. Capacity Leasing in Cloud Systems using the OpenNebula Engine. *Workshop on Cloud Computing and its Applications*, 2008.

[TMMVL12] Johan Tordsson, Ruben S. Montero, Rafael Moreno-Vozmediano, and Ignacio M. Llorente. Cloud brokering mechanisms for optimized placement of virtual machines across multiple providers. *Future Generation Computer Systems*, 28(2):358 – 367, 2012.

[Vou04] M.A Vouk. Cloud computing–issues, research and implementations. *Journal of Computing and Information Technology*, 16(4):235–246, 2004.

[VRMCL09] Luis M. Vaquero, Luis Rodero-Merino, Juan Caceres, and Maik Lindner. A Break in the Clouds: Towards a Cloud Definition. *ACM SIGCOMM Computer Communication Review*, 39:50–55, January 2009.

[Web15a] Green Cloud - The Green Cloud Simulator, `http://greencloud.gforge.uni.lu/`, June 2015.

[Web15b] GRIDSIM, Home Page, `http://www.cloudbus.org/gridsim/`, June 2015.

[Web15c] ICanCloud, Home Page, `http://www.arcos.inf.uc3m.es/~icancloud/Home.html`, June 2015.

[Web15d] SIMGRID, Cloud and Virtualization Contributions, `http://simgrid.gforge.inria.fr/contrib/clouds-sg-doc.html`, June 2015.

[Web15e] SIMGRID, Versatile Simulation of Distributed Systems, `http://simgrid.gforge.inria.fr/`, June 2015.

[WM14] Simon S. Woo and Jelena Mirkovic. Optimal application allocation on multiple public clouds. *Computer Networks*, 68(0):138 – 148, 2014. Communications and Networking in the Cloud.

[WNLL13] Wei Wang, Di Niu, Baochun Li, and Ben Liang. Dynamic cloud resource reservation via cloud brokerage. In *Proceedings of the 2013 IEEE 33rd International Conference on Distributed Computing Systems*, ICDCS '13, pages 400–409, Washington, DC, USA, 2013. IEEE Computer Society.

[YKA10] S Yi, D. Kondo, and A. Andrzejak. Reducing costs of spot instances via checkpointing in the amazon elastic compute cloud. *Cloud Computing, IEEE International Conference on*, 0:236–243, 2010.

[ZCB10] Qi Zhang, Lu Cheng, and Raouf Boutaba. Cloud computing: state-of-the-art and research challenges. *Journal of Internet Services and Applications*, 1:7–18, 2010.

[ZZZQ10] Minqi Zhou, Rong Zhang, Dadan Zeng, and Weining Qian. Services in the cloud computing era: A survey. In *Universal Communication Symposium (IUCS), 2010 4th International*, pages 40–46, Oct 2010.